3-29-2013

Fault and Performance Monitoring in Wireless LANs

Xian Chen
University of Connecticut - Storrs, xian.chen@engr.uconn.edu

Fault and Performance Monitoring in Wireless LANs

Xian Chen, Ph.D.

University of Connecticut, 2013

Wireless LANs (WLANs) have been widely deployed in enterprise and campus networks. With this wide deployment, it becomes increasingly important to understand the behavior of WLANs, and automatically manage WLANs to ensure their normal operation. In this dissertation work, we study three problems on fault and performance monitoring in WLANs. Specifically, we investigate wireless sniffer channel selection for WLAN monitoring, and network performance, IP address usage characteristics of smart mobile handhelds (MHDs) in a university campus WLAN.

Air sniffering is a widely-used and effective technique to monitor access points in WLANs. This technique, however, requires a large number of sniffers and generates a large amount of data. These challenges can be overcome by channel sampling, where each sniffer samples the network traffic by visiting multiple channels periodically. In the first part of the dissertation, we address an important problem in channel sampling, namely, how to select channels for sniffers to reduce monitoring cost. Specifically, we study two channel selection problems. Both of them require that each AP be monitored by at least one sniffer, and in addition, one problem minimizes the maximum number of channels that a sniffer listens to, while the other minimizes the total number of channels that the sniffers listen to. We propose three algorithms, one based on integer program, LP-relaxation, and greedy heuristic, to solve each problem. The performance of the algorithms is evaluated extensively using real-world traces.

Smart mobile handhelds (MHDs) such as smartphones have been adopted at a remarkable speed. Despite the recent flurry of research on various aspects of smart MHDs, little is known about their network performance in wireless LANs. The second part of the dissertation investigates the network performance of MHDs and limiting factors that affect the network performance in a university campus WiFi network. Our findings provide valuable insights on content distribution, server provisioning, MHD system design, and application-level protocol design.

The fast growth of MHDs poses challenges on managing IP address in campus WLANs. In the third part of this dissertation, we study two five-week long DHCP traces collected of two semesters from UConn WLAN, and analyze session length and IP address usage characteristics. We use both two and three stage hyper-exponential model session length (i.e. the duration that a user owns an IP address). In addition, we propose an analytical model to estimate the number of concurrent IP addresses in a WLAN. Evaluation results demonstrate our model is accurate. Our model can help network administrators to predict the demand of IP address in the network, and take proactive actions to satisfy future IP address demand.

Fault and Performance Monitoring in Wireless LANs

Xian Chen

B.E.,Beijing University of Aeronautics & Astronautics, 2003

M.E.,Beijing JiaoTong University, 2006

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2013

APPROVAL PAGE

Doctor of Philosophy Dissertation

Fault and Performance Monitoring in Wireless LANs

Presented by

Xian Chen, B.E., M.E.

Major Advisor _____
Bing Wang

Associate Advisor _____
Reda A. Ammar

Associate Advisor _____
Lester Lipsky

Associate Advisor _____
Jun-Hong Cui

Associate Advisor _____
Zhijie Shi

University of Connecticut

2013

ACKNOWLEDGEMENTS

My heartfelt gratitude goes to my major advisor, Professor Bing Wang, for her excellent guidance, continuous support, and generous encouragement throughout my Ph.D. study. I feel extremely lucky to be her Ph.D student, and have her as a constant source of wisdom, courage and skills. Her sharp insight and concise description of complicated research problems, down-to-earth attitude toward research, and her dedication have set an example of academic perfection, from which I will continue to benefit in my future career. I would like to thank Professor Sanguthevar Rajasekaran, and Professor Reda Ammar, and Professor Zhijie Shi, and Professor for their advice and broad knowledge. It was my great pleasure working with such intelligent and responsible professors. I would like to thank Professor Kyoungwon Suh in the Illinois State University, for his help on my work in networking performance analysis.

I would like to extend my gratitude to my colleagues, Zheng Guo, Wei Zeng, Yibo Zhu, Haining Mo, Yuan Song, for their discussions and comments in my research. My thanks also go to Jun Liu, Sixia Chen, Zhong Zhou, and all my friends, for their endless support whenever I needed. Last, but not least, I give my deepest gratitude and love to my parents and my wife, who have been a constant source of happiness and motivation for me. Their love and support have been my inspiration to accomplish the doctoral program. To my family, I dedicate this dissertation.

TABLE OF CONTENTS

Chapter 1

Background and Motivation

1.1 Introduction and Motivation

Large scale WLANs deployment can consist of thousands of WiFi stations with dozens or hundreds of access points. The performance WLANs may be degraded due to a number of reasons.

- *Limitation of radio signal*: wireless networks usually use radio waves for data communication. The strength of the radio signal limits the transmission range. In addition, radio strength fades due to multipath propagation or obstacles. The limited transmission range can cause hidden terminal problem, which results in message collisions and transmission failures. Signal fading can cause packet loss or high communication latency which results in poor network performance.

- *Interference (collisions)*: interference exists because nodes may compete for shared wireless medium when sending packets. It usually results in unsuccessful transmission and packet drop, that cause lower data throughput, leading to degraded network performance.

- *Malicious usage*: unauthorized clients or access points can cause addition competition of the shared medium. It not only brings security challenges but also degrades the performance.

In the last decade, Wireless LANs (WLANs) have become ubiquitous in both enterprise and campus networks. Traditionally, wireless hosts are predominantly laptops. Recently, smart mobile handhelds (MHDs) such as smartphones have been adopted at a remarkable speed. MHD is the trend of unified communications that integrate telecom and Internet services onto a single device because it has combined the portability of cell-phones with the computing and networking power of laptops. They have been used for a wide range of applications including surfing web, checking email, watching video, accessing social networking services, and online games. MHDs use WLANs whenever available because of lower delay and energy consumption compared with the cellular interface.

Compared to laptops, MHDs are less powerful and more resource restrained. The network performance of MHDs are affected by many factors range from application level down to physical level. The proliferation of MHDs have posed stress on current wireless network infrastructure. Furthermore, the frequent network association from MHDs have resulted an increase in the network IP addresses usage.

1.2 Fault and Performance Monitoring of WLANs

As stated before, the performance of WLANs can be degraded due to several reasons. Existing solutions for monitoring the run-time status and performance of WLAN fall into three categories. The first category is to use SNMP protocol to collect the access points running information [15, 16, 25, 26, 45, 77]. This approach, however, cannot collect MAC and PHY level information (e.g., signal strength, spectrum density, collision, retransmissions, and backoff times), which are critical for investigating the root causes of the performance degradation. In the second type of approaches, all WiFi stations are responsible to store history of their own activities, which are then transferred to a central server following a certain schedule. This approach requires further modification of existing driver and softwares. Furthermore, not all WiFi stations are equipped with sufficient storage capacity. The third category is air sniffing, where a set of sniffers (also called air monitors, wireless monitors, or radio monitors) are placed inside a WLAN , each passively listening to the air waves in its vicinity, and collecting detailed MAC/PHY information (e.g., [15,16,20,24,25,45,46,60,62,64,77], more details in Section 2.2). Air sniffing has been shown to complement wire side monitoring that uses SNMP and base-station logs. However, large scale air sniffing deployment faces several challenges. Firstly, it requires a large number of sniffers to monitor all of the access points. Secondly, the sniffers generate a large amount of measurement data, which can be expensive to store, transfer and process. To overcome these challenges, *channel sampling* is introduced. This methodology requires the sniffers to sample the network traffic by visiting multiple channels periodically [29].

1.3 Performance Monitoring of MHDs in WLANs

The emergence of MHDs poses challenges to the performance monitoring of WLANs. Despite the recent flurry of research on various aspects of smart MHDs, little is known about their network performance in wireless LANs. Consider the rapid growth of MHDs, understanding the network performance of MHDs is critical to the WLAN management as well. There are numbers of approaches that are applied to analyze the performance of MHDs. One approach is using native application to monitor and record the traffic on MHDs [33, 41, 68]. Then using offline tools to analyze the traffic patterns, performance, and mobility from the traces. However, this approach mixes the cellular and WiFi traffic, making it hard to understand the characteristics of MHDs inside WiFi networks. Furthermore, this approach leads to privacy concerns, and it is difficult to collect large scale traces.

The second approach overcomes the shortages of the first one. This approach collects the network level traces by placing a monitor point on the WLAN main gateway router [9,21,72,73]. Therefore, this monitor point could record both incoming and outgoing traffic from the devices within the WLAN. Clearly, the traces from this approach contain large numbers of users, and thus the conclusions are more representative.

1.4 modeling MHD characteristics in WLANs

Compared to wireless non-handheld devices (NHDs) such as Windows laptops and MacBooks, MHDs are being adopted at a much faster pace. In addition, since MHDs are smaller and easier to carry, users tend to use these devices to access the network much more often than the NHDs. These characteristics of MHDs leads to different network usage characteristics, such as session length and IP address demand. The study of

session length of network clients is important in various aspects. For instance, the understanding of session length is critical for network modeling and mobility study [23,70]. In addition, this can be beneficial in administration, capacity planning and deployment of wireless infrastructures, protocol design for wireless applications and services, and their performance analysis. Access points, proxies, and servers can use the estimation of their clients' session length to prepare the handoff, share clients or traffic load with each other, and ensure a better service quality [52]. The existing session length study works are focus on NHDs, which have different features than that of MHDs. Moreover, the existing session length modeling requires extensive computation. Compared to the NHDs, more frequent accessing the network from MHDs means more workload to the DHCP servers. Furthermore, the explosion of the number of MHDs in the wireless LANs also introduce more demands on the IP addresses. Therefore, studying the MHDs IP usage behavior has become a critical task for the WLAN management. In Chapter 4, we analyze two five-weeks long DHCP WLAN traces and provide analytical models to study the MHDs' IP address usage characteristics. Our models are easier to compute and

1.5 Contributions of This Dissertation

The contributions of this dissertation are three-fold: (1) leveraging sniffer channels selecting algorithms for WLAN monitoring, (2) analyzing large scale campus WLAN traffic to find the limiting factors that could affect the network performance of MHDs, and (3) analyzing large scale DHCP traces to obtain the MHDs IP address usage characteristics and propose a modeling approach to predict the concurrent number of users in the network.

First, we address an important problem in channel sampling, namely, how to select channels for the sniffers to reduce the monitor cost. More specifically, we consider two problems. Both of them require that each AP be monitored by at least one sniffer, and in addition, one problem minimizes the maximum number of channels that a sniffer listens to, while the other minimizes the total number of channels that the sniffers listen to. Both optimization problems aim at reducing the number of channels that a sniffer needs to scan (in terms of worst case and average sense, respectively) since when a sniffer scans less channels, it can spend more time on each channel to improve sampling quality. We prove that both optimization problems are NP-hard. For each problem, we propose three algorithms to solve it, one based on integer programming (IP), one based on LP-relaxation, and the third based on a greedy heuristic. We evaluate the performance of the various algorithms using two real-world datasets. Our results show that, for each problem, all the three algorithms are effective in achieving their optimization goals, and overall, LP-based algorithms outperform the other two algorithms.

Secondly, we measure the performance of smart MHDs inside a university campus WiFi network. Specifically, our study is over a data set that is passively captured by a monitor placed at a gateway router in the University of Connecticut (UConn). The data set is collected over three days (2.9TB of data), containing traffic from various wireless devices, including MHDs such as iPhones, iPod touches, iPads, Android phones, Windows phones, and Blackberry phones, and wireless non-handheld devices (NHDs) such as Windows laptops and MacBooks. Analyzing the data set, we find HTTP is the dominant traffic type, accounting for over 92% of the TCP flows. We therefore focus on the performance of HTTP flows in this paper. The behavior of HTTP is complicated: we find many HTTP flows contain multiple HTTP requests,

and a significant portion of an HTTP flow is idling (with no data packets). Hence, the traditional throughput metric (i.e., the amount of data downloaded in a flow divided by the total duration of the flow) may introduce bias in measuring the performance of HTTP flows. We therefore define a metric, *per-flow servicing rate*, i.e., the amount of data downloaded corresponding to HTTP requests in a flow divided by the downloading duration of the flow, to quantify the performance of an HTTP flow (see Section 3.4). This metric is interesting in its own right: it represents the network performance of an HTTP flow, while excluding the effect of various delays (e.g., client processing delays and user pause times) that are irrelevant to network performance. We make the following main findings: (1) Compared to non-handheld devices (NHDs), MHDs use well provisioned Akamai and Google servers more heavily, which boosts the overall network performance of MHDs. Furthermore, MHD flows, particularly short flows, benefit from the large initial congestion window that has been adopted by Akamai and Google servers. (2) MHDs tend to have larger local delays inside the WiFi network and are more adversely affected by the number of concurrent flows. (3) Earlier versions of Android OS (before 4.X) cannot take advantage of the large initial congestion window adopted by many servers. On the other hand, the large receive window adopted by iOS is not fully utilized by most flows, potentially leading to waste of resources. (4) Some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks.

Last, we analyze two five-week long traces collected in two semesters from UConn WLAN. We propose a model to describe the session length and IP allocation length distributions. Furthermore, we develop a model to predict the number of concurrent users (and hence the demand on IP address) at each time point in the wireless LAN. Although motivated by MHDs, our model is also applicable to NHDs. Evaluation using

the two traces demonstrates that our model is accurate. More specifically, the average difference of the predicted value from the model and the actual value is between 8% to 12%. Using our model, network administrators can predict the demand on IP addresses in a wireless LAN, and take proactive actions to satisfy future IP address demands.

1.6 Dissertation Roadmap

The remainder of this dissertation is organized as follows. In Chapter 2, we describe our work on sniffer channel selection for monitoring wireless LANs. We first present the motivation of solving sniffer channel selection problem in Section 2.1 Then we review related work in Section 2.2. Section 2.3 describes the problem setting. After that, Sections 2.4 and 2.5 describe our sniffer channel selection algorithms for the two optimization problems, respectively. Section 2.6 presents performance evaluation. Finally, we summarize our work in Section 2.7.

In Chapter 3, we present our work on network performance of smart mobile handhelds in a university campus WiFi network. We first review the challenges for analyzing the network performance of MHDs in Section 3.1. Section 3.2 describes related work. Section 3.3 describes data collection and classification. Section 3.4 introduces the performance metric. Section 3.5 describes our methodology. Section 3.6 presents network performance of MHDs, and explores the impact of network and application layer factors on the performance. Section 3.7 discusses the applicability of the findings to other networks. Last, we summarize our work Section 3.8.

In Chapter 4, we characterize session length and model IP address usage by smart MHDs in wireless LANs. We first discuss the challenges and motivation of our study in Section 4.1. We next presents background on DHCP in Section 4.2. Section 4.3 describes our data set and methodology. Section 4.4 describes modeling session length.

Section 4.5 presents IP address usage statistics and model the concurrent number of IP addresses. Section 4.6 discusses two applications of applying our model to predict the concurrent number of users in the network. Finally, Section 4.7 summarizes our work.

Finally, we conclude this dissertation and present future work in Chapter 5.

Chapter 2

Sniffer Channel Selection for Monitoring Wireless LANs

2.1 Introduction

Wireless LANs (WLANs) have been widely deployed in enterprise and campus networks. With this wide deployment, it becomes increasingly important to understand the behavior of WLANs, and automatically manage WLANs to ensure their normal operation and security. A widely-used and effective technique for understanding and monitoring WLANs is air sniffing, where a set of sniffers (also called air monitors, wireless monitors, or radio monitors) are placed inside a WLAN , each passively listening to the air waves in its vicinity, and collecting detailed MAC/PHY information (e.g., [15, 16, 20, 24, 25, 45, 46, 60, 62, 64, 77], more details in Section 2.2). Air sniffing has been shown to complement wire side monitoring that uses SNMP and base-station logs [15,16,25,26,45,77]. This is because the detailed MAC/PHY information (e.g., signal strength, spectrum density, collision, retransmissions, and backoff times) provides valuable insights into the behavior of wireless medium and protocols, which can help

network administrators to optimize radio coverage and determine the root causes of network faults for effective trouble shooting. In addition, for mission critical WLANs with high security requirements, such as those deployed in banks or military bases [1], PHY/MAC visibility provided by the sniffers is critical, as wired solution can only detect upper layer threats.

While requiring additional infrastructure, the insights and benefits achieved air sniffing cannot be achieved by traditional monitoring techniques (e.g., SNMP). On the other hand, large-scale WLAN monitoring through air sniffing faces several challenges. First, it requires a large number of sniffers, which can be costly to deploy and difficult to manage. This problem is compounded by the fact that access points (APs) in WLANs can operate on different channels (e.g., 802.11b/g supports 3 orthogonal channels, and 802.11a supports 13 orthogonal channels), while an air sniffer can only listen to a single channel at a given point of time (although a sniffer may use multiple radios to monitor multiple channels simultaneously, such type of sniffers are large and expensive to deploy [29]). Therefore, in the worst case, the required number of sniffers can be the same as the number of APs. Secondly, the sniffers generate a large amount of measurement data, which can be expensive to store, transfer and process. For instance, in [25], up to 80 Mbps of traffic is generated for monitoring an academic building, which needs to be transferred and processed at a central server.

The above challenges in large-scale air sniffing can be overcome by *channel sampling*, where each sniffer samples the network traffic by visiting multiple channels periodically [29]. Using channel sampling, a sniffer can monitor multiple nearby APs that operate on different channels, and hence less sniffers are needed. Furthermore, traffic sampling leads to less amount of measurement data. As shown in [29], although not

capturing all the traffic, channel sampling is useful for a number of applications, including security monitoring, anomaly detection, fault diagnosis, network characterization, and assistance to AP deployment.

In this paper, we address an important problem in channel sampling, namely, how to select channels for the sniffers to reduce the monitor cost. More specifically, we consider two problems. Both of them require that each AP be monitored by at least one sniffer, and in addition, one problem minimizes the maximum number of channels that a sniffer listens to, while the other minimizes the total number of channels that the sniffers listen to. Both optimization problems aim at reducing the number of channels that a sniffer needs to scan (in terms of worst case and average sense, respectively) since when a sniffer scans less channels, it can spend more time on each channel to improve sampling quality[1] . We prove that both optimization problems are NP-hard. For each problem, we propose three algorithms to solve it, one based on integer programming (IP), one based on LP-relaxation, and the third based on a greedy heuristic. We evaluate the performance of the various algorithms using two real-world datasets. Our results show that, for each problem, all the three algorithms are effective in achieving their optimization goals, and overall, LP-based algorithms outperform the other two algorithms.

The rest of the paper is organized as follows. Section 2.2 describes related work. Section 2.3 describes the problem setting. Sections 2.4 and 2.5 describe our sniffer channel selection algorithms for the two optimization problems, respectively. Section 2.6 presents performance evaluation. Finally, Section 2.7 concludes the paper and describes future work.

[1]We discuss tradeoffs of these two optimization problems in Section 2.3. In practice, a network administrator may choose to use one of these two optimization objectives based on the goals of the WLAN monitoring.

2.2 Related work

Several studies use air sniffing to understand and/or manage WLANs. Adya *et al.* [12] propose a client-based architecture that instruments wireless clients and (if possible) APs to monitor wireless medium and their nearby devices to detect and diagnose faults. Bahl *et al.* [17] propose using dense array of inexpensive radios (DAIR) through USB wireless adaptors that are attached to desktops to detect rogue wireless devices and Denial of Service attacks on WLANs. Later on, Chandra *et. al.* extend this DAIR architecture to incorporate location estimation and develop a location-based management system for WLANs [20]. Yan and Chen propose a model-based fault diagnosis approach that detects and localizes faults through self-monitoring at the APs [75]. Yeo *et al.* propose a framework that merges link-level measurement from multiple distributed air sniffers for WLAN management [77,78]. This framework is substantially extended in Jigsaw [25] and Wit [46], where the authors provide formal and systematic techniques to construct a global view of the network by merging and synchronizing traces from multiple locations. The global view has been used for understanding many aspects of WLANs, including congestion [45], link-layer losses and anomalies [46,77], co-channel interference [25], and sources of delays [24]. It has also been used to determine root-cause of physical-layer anomalies [62], and identify threats and attacks [60, 61]. In addition to the above studies in academia, air sniffing has also been used in many commercial products (e.g., [2–4,7,11]). Our study uses air sniffers to monitor APs, and focuses on sniffer channel assignment, which has not been studied in these literatures.

As described earlier, air sniffing through dedicated sniffers can lead to high deployment cost and a large amount of monitoring traffic. The studies of [29, 30] propose

channel sampling to address the above two issues. In particular, the study of [29] proposes two sampling strategies, equal-time sampling where a sniffer spends equal amount of time scanning each channel, and proportional sampling where the amount of time that a sniffer spends on a channel is proportional to the amount of traffic on that channel. These two strategies are improved in [30] where the scanning of the sniffers are coordinated to increase the number of unique frames. Our study determines the set of channels that a sniffer scans during channel sampling. We require each sniffer to monitor a *subset* of selected channels, while [29, 30] require each sniffer to monitor all available channels, regardless of whether the channels are being used or not by the nearby APs[2] . By eliminating the scanning over unused channels, our approach provides more effective traffic sampling. Several recent studies design centralized or distributed sniffer channel assignment algorithms [15, 16, 63, 64]. These studies assume that the sniffers have multiple radios or assign channels to sniffers in a probabilistic manner, which differ from the context of channel sampling as in our study.

2.3 Problem setting

We now describe the problem setting. The key notation is summarized in Table 1 for easy reference. Consider a WLAN with a set of APs, V. Each AP uses a single radio, and hence a single channel, at any point of time (if an AP uses multiple channels simultaneously, we can regard it as multiple APs, each with a single channel). Let C denote the set of channels that the APs operate on. In particular, suppose AP v operates on channel c_v, $c_v \in C$. A set of sniffers (or monitors), M, is spread out

[2]One motivation of scanning all the channels in [29, 30] is that it can capture rogue APs that operate on unused channels. Rogue APs, however, can be effectively detected using other approaches such as [74, 76].

Table 1: Key notation.

Notation	Definition
V	set of APs
C	set of channels that the APs use
c_v	channel that AP v uses, $v \in V, c_v \in C$
M	set of sniffers
M_v	set of sniffers that can hear the transmission from AP v
$\varphi(v)$	assignment to AP v (the set of sniffers that monitor v)
$C_\varphi(m)$	set of channels that sniffer m listens to based on assignments $\varphi(\cdot)$
M_φ	set of sniffers that are used based on assignments $\varphi(\cdot)$

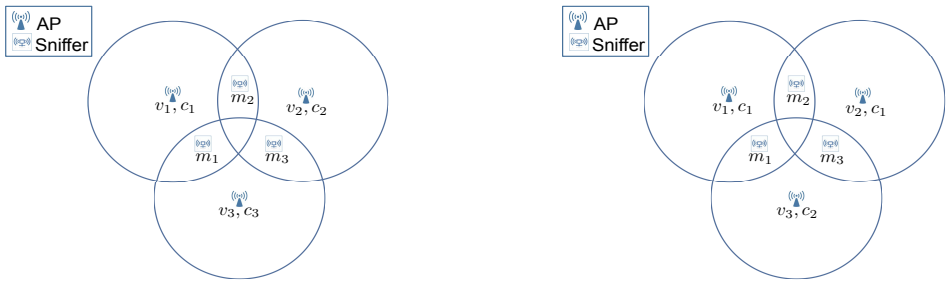


Figure 1: Two examples to illustrate the problem setting. Both examples contain three APs, v_1, v_2 and v_3, and three sniffers, m_1, m_2, and m_3, where sniffer m_1 can monitor v_1 and v_3; sniffer m_2 can monitor v_1 and v_2; and sniffer m_3 can monitor v_2 and v_3. In (a) APs v_1, v_2, and v_3 use channels c_1, c_2, c_3, respectively; in (b) APs v_1 and v_2 use channel c_1, and AP v_3 uses channel c_2.

in the WLAN to monitor the APs[3] . Let M_v denote the set of sniffers that are within the transmission range of v (i.e., M_v is the set of sniffers that can overhear the transmission of v when listening to channel c_v), $M_v \subseteq M$. We assume that $|M_v| \geq 1$, i.e., at least one sniffer can monitor v, $\forall v \in V$. Each sniffer has a single radio, and switches among multiple channels to monitor its nearby APs when these APs operate on different channels.

[3]The sniffers can be deployed as a separate infrastructure, or integrated on the APs themselves as in [75]. In this paper, for ease of exposition, we assume sniffers are deployed as a separate infrastructure.

Motivated by what is adopted by commercial products (e.g., [7]), we assume that the WLAN uses a centralized management architecture, where a central controller manages the operation of the APs. The central controller knows the coordinates of the APs, and determines the channel for each AP. Furthermore, it knows the location of the sniffers, and determines the set of channels that each sniffer scans based on the locations of the APs and sniffers, and the channels of the APs. The PHY/MAC information collected by the sniffers is transmitted to the central controller for fault diagnosis and security analysis. This centralized architecture has many benefits: it reduces deployment and operating expenses, and significantly simplifies daily operation and management of small to large-scale WLANs. Fig. 1 illustrates the problem setting using two examples. Both examples contain three APs, v_1, v_2, v_3, and three sniffers, m_1, m_2, m_3, that are controlled by the central controller. In addition, sniffer m_1 is in the transmission ranges of v_1 and v_3; sniffer m_2 is in the transmission ranges of v_1 and v_2; sniffer m_3 is in the transmission ranges of v_2 and v_3. They differ in the channels that the APs use: in Fig. 1(a), APs v_1, v_2, and v_3 use channel c_1, c_2, c_3, respectively, while in Fig. 1(b), APs v_1, v_2 use channel c_1, and v_3 uses channel c_2.

Our goal is to determine the set of channels that each sniffer monitors. Let $\varphi(v)$ denote the set of sniffers that monitor AP v, referred to as *assignment* to v. Let $C_\varphi(m)$ denote the set of channels that sniffer m monitors based on the assignment $\varphi(\cdot)$. Then $C_\varphi(m) = \{c_v \mid m \in \varphi(v)\}$. Clearly, $C_\varphi(m) = \emptyset$, if $m \notin \varphi(v), \forall v \in V$. In this case, sniffer m is not used, and does not need to be deployed. We further define the *workload* of a sniffer as the number of channels that the sniffer scans. A sniffer is used if it monitors at least one channel, i.e., its workload is non-zero. Let $M_\varphi \subseteq M$ denote the set of sniffers that are being used. That is, $M_\varphi = \{m \mid C_\varphi(m) \neq \emptyset\}$.

We consider two sniffer channel selection problems. Both variants require that each AP be monitored by at least one sniffer, i.e., $\varphi(v) \neq \emptyset, \forall v \in V$. In addition, the first variant minimizes the maximum number of channels that a sniffer listens to (i.e., minimizes $\max_{m \in M} |C_\varphi(m)|$), while the second variant minimizes the sum of the channels that the sniffers listen to (i.e., minimizes $\sum_{m \in M} |C_\varphi(m)|$). We refer to these two variants as *min-max* and *min-sum sniffer channel selection problems*, respectively. In the min-max problem, the workloads of the sniffers are more balanced than those in the min-sum problem. On the other hand, the min-sum problem may need to use less sniffers and hence may have a lower deployment cost. The intuition comes from a special case: when there is a single channel, the min-sum problem minimizes the number of sniffers that need to be used. This is because, in this case, each sniffer needs to scan at most one channel, and hence minimizing the sum of the channels is equivalent to minimizing the number of sniffers that are used. Indeed, our extensive simulation demonstrates that the min-sum problem generally needs less sniffers than the min-max problem (see Section 2.6).

We further illustrate the difference of the min-max and min-sum problems using the two examples in Fig. 1. The optimal solutions of the two problems are the same for the example in Fig. 1(a), while differ for the example in Fig. 1(b). Specifically, for the example in Fig. 1(a), the optimal channel selection for both problems is making sniffers m_1, m_2, m_3 listen to channels c_1, c_2, c_3, respectively. For the example in Fig. 1(b), the optimal solution of the min-max problem is one, i.e., m_1 uses channel c_1 to monitor v_1, m_2 uses channel c_1 to monitor v_2, and m_3 uses channel c_2 to monitor v_3, which requires three sniffers, and the sniffer workloads are balanced. This channel selection leads to a solution of three for the min-sum problem, which is not optimal. The optimal solution of the min-sum problem is two, e.g., achieved as m_2 uses channel c_1 to monitor APs

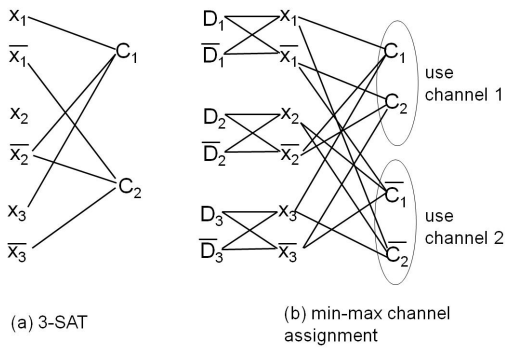


Figure 2: Illustration of the reduction from 3-SAT to the min-max sniffer channel assignment problem.

v_1 and v_2, and m_3 uses channel c_2 to monitor v_2, which only requires two sniffers, but the sniffer workloads are not balanced.

Last, neither optimization problem explicitly minimizes the number of sniffers that is being used (i.e., $|M_\varphi|$). Therefore, after solving the optimization problems, we post process the assignments to remove redundant sniffers to reduce the number of sniffers that are being used (see Sections 2.4 and 2.5).

2.4 Algorithms for min-max sniffer channel selection

We first prove that the min-max sniffer channel selection problem is NP-hard by reducing 3-SAT (a known NP-complete problem) to it.

Proof. Let ϕ be an instance of 3-SAT problem. Suppose ϕ contains n variables, $x_1, \ldots, x_n$, $2n$ literals, $x_1, \bar{x_1}, \ldots, x_n, \bar{x_n}$, and m clauses, $C_1, \ldots, C_m$. The corresponding sniffer channel assignment problem contains $2n$ sniffers, $x_1, \bar{x_1}, \ldots, x_n, \bar{x_n}$, and $(2m{+}2n)$ APs, $C_1, \bar{C_1}, \ldots, C_m, \bar{C_m}$, and $D_1, \bar{D_1}, \ldots, D_n, \bar{D_n}$. Each sniffer can operate on two channels, 1 and 2. Each AP operates on one channel. In particular, AP C_i operates on

channel 1, and AP $\bar{C}_i$ operates on channel 2, $i = 1, \ldots, m$; AP D_i operates on channel 1, and AP $\bar{D}_i$ operates on channel 2, $i = 1, \ldots, n$. If x_i is used in clause C_j in ϕ, then in the corresponding sniffer channel assignment problem, sniffer x_i can monitor AP C_j using channel 1, and sniffer $\bar{x}_i$ can monitor AP $\bar{C}_j$ using channel 2. Similarly, if $\bar{x}_i$ is used in clause C_j in ϕ, then in the corresponding sniffer channel assignment problem, sniffer $\bar{x}_i$ can monitor AP C_j using channel 1, and sniffer x_i can monitor AP $\bar{C}_j$ using channel 2. Last, we allow and only allow sniffers x_i and $\bar{x}_i$ to monitor APs D_i and $\bar{D}_i$.

Fig. 2 shows an example illustrating the relationship between ϕ and the corresponding sniffer channel assignment problem. In the example, $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$. We represent the relationship between the literals and the clauses in Fig. 2(a), where a clause is connected to a literal if it contains that literal. Fig. 2(b) shows the corresponding sniffer channel assignment problem. For instance, Fig. 2(a) shows that x_1 is used in clause C_1 in ϕ. Correspondingly, Fig. 2(b) shows that sniffer x_1 uses channel 1 to monitor AP C_1, and sniffer $\bar{x}_1$ uses channel 2 to monitor AP $\bar{C}_1$. For ease of illustration, in Fig. 2(b), the upper circle contains $C_1, \ldots, C_m$ that use channel 1; and the lower circle contains $\bar{C}_1, \ldots, \bar{C}_m$ that use channel 2. We further have x_i connects to D_i and $\bar{D}_i$, and $\bar{x}_i$ connects to D_i and $\bar{D}_i$.

We next show that there is a satisfying assignment to ϕ iff the solution to the min-max sniffer channel assignment problem is 1, i.e., each sniffer needs to scan at most one channel. We first prove that when ϕ is satisfiable, then each sniffer needs to scan at most one channel. Suppose that there exists an assignment to x_i, $i = 1, \ldots, n$, so that all the clauses in ϕ are true, and hence ϕ is true. Consider an arbitrary clause C_j. Since C_j is true, at least one literal used in C_j must be true. We consider the following two cases:

- Case 1. Suppose a literal in C_j, x_i, is true. Then in the sniffer channel assignment problem, we let sniffer x_i monitor channel 1 (and hence it can monitor AP C_j), and sniffer $\bar{x_i}$ monitor channel 2 (and hence it can monitor AP $\bar{C_j}$). Under this channel assignment, both APs C_j and $\bar{C_j}$ are monitored.

- Case 2. Suppose a literal $\bar{x_k}$ in C_j is true (i.e., x_k is false). Then in the sniffer channel assignment problem, we let sniffer x_k monitor channel 2, and sniffer $\bar{x_k}$ monitor channel 1. Then again both APs C_j and $\bar{C_j}$ are monitored.

Summarizing the above two cases, we can find sniffer channel assignment for x_i, $i = 1, \ldots, n$, so that all C_j's and $\bar{C_j}$'s are monitored, and each sniffer needs to monitor at most one channel. For APs D_i and $\bar{D_i}$, since only x_i and $\bar{x_i}$ are allowed to monitor them, and x_i and $\bar{x_i}$ monitor two different channels, it is easy to see that both of them can be monitored based on the current sniffer channel assignments. More specifically, if x_i monitors channel 1 (i.e., $\bar{x_i}$ monitors channel 2), we let x_i monitor D_i and $\bar{x_i}$ monitor $\bar{D_i}$ (since D_i and $\bar{D_i}$ operate on channels 1 and 2, respectively); if x_i monitors channel 2, we let x_i monitor $\bar{D_i}$ and $\bar{x_i}$ monitor D_i, $i = 1, \ldots, n$. In summary, when ϕ is satisfiable, we can find channel assignments to all the sniffers so that each sniffer needs to monitor at most one channel and all the APs are monitored. Therefore, the solution to the min-max problem is 1.

We now prove that if each sniffer needs to scan at most one channel in the min-max sniffer channel assignment problem, then ϕ is satisfiable. Suppose there exists a channel assignment to all the sniffers, so that all the APs are monitored and each sniffer monitors at most one channel. Consider all the sniffer pairs $(x_k, \bar{x_k})$, $k = 1, \ldots, n$. Let $S_{i,j}$ denote the set of sniffer pairs in which sniffer x_k scans channel i and $\bar{x_k}$ scans channel j, $i, j = 1, 2$. We then have $S_{1,1} = S_{2,2} = \emptyset$. This is because, since D_i and $\bar{D_i}$

operate on two different channels and both of them are monitored, and in addition only x_i and $\bar{x}_i$ are allowed to monitor them, we must have x_i and $\bar{x}_i$ monitor two different channels, $i = 1, \ldots, n$. Hence we have $S_{1,1} = S_{2,2} = \emptyset$. In this case, for the APs to be monitored, we need $S_{1,2} \neq \emptyset$ and/or $S_{2,1} \neq \emptyset$. Consider an arbitrary sniffer pair $(x_i, \bar{x}_i)$. Then we have the following two scenarios:

- Sniffers x_i and $\bar{x}_i$ monitor channels 1 and 2, respectively. Then in ϕ, we set x_i to be true and set $\bar{x}_i$ to be false. Since x_i monitors channel 1, it can only monitor APs that use channel 1 (i.e., APs in the upper circle in Fig. 2(b)). Suppose x_i monitors AP C_j. Then by the mapping between the 3-SAT and the min-max problem, x_i is used in clause C_j in ϕ, and therefore C_j is satisfied.

- Sniffers x_i and $\bar{x}_i$ monitor channels 2 and 1, respectively. Then in ϕ, we set the x_i to be false and set $\bar{x}_i$ to be true. Following a similar argument as before, suppose $\bar{x}_i$ monitors AP C_j, then by the mapping between the 3-SAT and the min-max problem, $\bar{x}_i$ is used in clause C_j in ϕ, then C_j is satisfied.

Summarizing the above two scenarios, since the sniffers in $S_{1,2} \cup S_{2,1}$ monitor all the APs, all the clauses in ϕ are satisfied. Hence we conclude that ϕ is satisfiable. Therefore, we have proved that if each sniffer needs to monitor at most one channel in the sniffer channel assignment problem, then there exists a satisfying assignment for ϕ, and hence complete the proof. □

In the following, we develop three algorithms to solve it. These three algorithms are based on integer programming (IP), linear programming (LP), and a greedy heuristic, referred to as *IP-min-max*, *LP-min-max*, and *Greedy-min-max*, respectively. We next describe the three algorithms in detail, and illustrate the results of solving the example in Fig. 1(a) (all three algorithms provide the same solution for the example in Fig. 1(b)).

After obtaining a solution using one of the three algorithms, some redundant sniffers may be removed while still satisfying all the constraints. We therefore also propose a post-processing procedure that removes redundant sniffers and is applicable to all the three algorithms at the end of this section.

2.4.1 IP-min-max

Let $x_{m,c}$ be a 0-1 variable. In particular, $x_{m,c} = 1$ denotes that sniffer m monitors channel c, and $x_{m,c} = 0$ denotes otherwise. Then the min-max sniffer channel selection problem can be formulated as an IP problem:

$$\text{minimize :} \quad \max_{m \in M} \sum_{c \in C} x_{m,c} \tag{1}$$

$$\text{subject to:} \quad \sum_{m \in M_v} x_{m,c_v} \geq 1, \forall v \in V \tag{2}$$

$$x_{m,c} \in \{0, 1\} \tag{3}$$

In the objective function, $\sum_{c \in C} x_{m,c}$, is the total number of channels that sniffer m listens to, and Constraint (2) denotes that each AP is monitored by at least one sniffer.

For a small-scale problem, the above IP problem can be solved directly (e.g., using CPLEX [8]) to obtain an optimal solution for the min-max problem. Afterwards, we determine the assignment for each AP, and the set of channels for each sniffer as follows. For AP v, let $\varphi(v) = \{m \mid m \in M_v, x_{m,c_v} = 1\}$. That is, we let all the sniffers that can overhear v and monitors c_v (i.e., the channel that v operates on) to monitor v. Correspondingly, we determine the set of channels that each sniffer monitors, i.e., $C_\varphi(m) = \{c \mid x_{m,c} = 1\}$, $\forall m$.

We now illustrate the results from IP-min-max using the example in Fig. 1(a). Solving the IP problem, we have $x_{m_i,c_i} = 1, i = 1, 2, 3$; others are 0. Leading to

an optimal solution of 1 for the min-max problem. For AP $v_i, i = 1, 2, 3$, we have $\varphi(v_i) = \{m_i\}$. And we have $C_\varphi(m_i) = c_i, i = 1, 2, 3$.

2.4.2 LP-min-max

For large problems, we may not be able to solve the IP problem in Section 2.4.1 directly. In our second algorithm, LP-min-max, we relax the integer constraint on $x_{m,c}$, and let $y_{m,c} \in [0, 1]$ be the relaxed value of $x_{m,c}$. The original IP problem then becomes an LP problem, which can be solved in polynomial time. After solving the LP problem, we choose channels based on $y_{m,c}$ as described Algorithm 1. In this algorithm, line 1 initializes $\varphi(v)$, $C_\varphi(m)$ and M_φ to empty sets, $\forall v \in V$, $\forall m \in M$. Let S_v represent the set of APs that have already been considered. Line 2 initializes S_v to an empty set. The algorithm then considers all the APs. For an AP $v \in V$, if one monitor, m, has already been selected to be used and can hear v (i.e., $m \in M_\varphi \cap M_v$), and furthermore m has been assigned to monitor c_v (i.e., $c_v \in C_\varphi(m)$), then we simply assign m to monitor v. If there are multiple such monitors, all of them are recorded in $\varphi(v)$. If no such monitor exists, we pick a sniffer, m, that leads to the maximum y_{m,c_v} among all the sniffers that are in the transmission range of AP v (line 8). Once we add m to M_φ, the APs that have already been considered may also be monitored by m. Lines 12-16 find such APs, and assign m to monitor them as well. Line 18 updates S_v.

We next briefly describe the complexity of LP-min-max. The LP problem can be solved in polynomial time. In particular, when using interior point method, the running time is $O((|M||C|)^4)$, where $|M||C|$ is the number of variables. After solving the LP problem, the running time of Algorithm 1 to assign sniffer channels is $O(|V|)$. Therefore, the complexity of LP-min-max is $O((|M||C|)^4 + |V|) = O((|M||C|)^4)$.

Algorithm 1 LP-min-max

$\varphi(v) = \emptyset, \forall v \in V$, $C_\varphi(m) = \emptyset, \forall m \in M$, $M_\varphi = \emptyset$
$S_v = \emptyset$
for all $v \in V$ **do**
  for all $m \in M_\varphi \cap M_v$ and $c_v \in C_\varphi(m)$ **do**
    $\varphi(v) = \varphi(v) \cup \{m\}$
  end for
  if $\varphi(v) = \emptyset$ **then**
    pick $m = \arg\max_{m \in M_v} y_{m,c_v}$
    $C_\varphi(m) = C_\varphi(m) \cup \{c_v\}$
    $\varphi(v) = \{m\}$
    $M_\varphi = M_\varphi \cup \{m\}$
    for all $v' \in S_v$ **do**
      if $m \in M_{v'}$ and $c_{v'} = c_v$ **then**
        $\varphi(v') = \varphi(v') \cup \{m\}$
      end if
    end for
  end if
  $S_v = S_v \cup \{v\}$
end for
Return $(\varphi, C_\varphi, M_\varphi)$

We now illustrate LP-min-max using the example in Fig. 1(a). Solving the LP problem, we have one solution, $y_{m_1,c_1} = 0.5$, $y_{m_1,c_3} = 0.5$, $y_{m_2,c_1} = 0.5$, $y_{m_2,c_2} = 0.5$, $y_{m_3,c_2} = 0.5$, $y_{m_3,c_2} = 0.5$. Based on the values of $y_{m,c}, \forall m \in M, c \in C$, we assign m_2 to monitor v_1 since both m_1 and m_2 can hear v_1, $y_{m_1,c_1} = y_{m_2,c_1}$[4] . Similarly, in the next iteration, we can assign m_2 to monitor v_2. Finally, we choose m_3 to monitor v_3. Therefore, we have a solution from LP-min-max is $C_\varphi(m_2) = \{c_1, c_2\}$ (m_2 uses channel c_1 and c_2), and $C_\varphi(m_3) = \{c_3\}$. Leading to a suboptimal solution of 2 for the min-max problem. The assignment results are $\varphi(v_1) = \{m_2\}$, $\varphi(v_2) = \{m_2\}$, and $\varphi(v_3) = \{m_3\}$. Since $C_\varphi(m_1) = \emptyset$, m_1 is not used.

Last, the following theorem states an approximation-ratio result for LP-min-max.

[4]Here we break ties arbitrarily. Developing other approaches for breaking ties is left as future work.

Theorem 1. *LP-min-max is an $O(r)$-approximation algorithm for the min-max sniffer channel selection problem, where $r = \max_{v \in V} |M_v|$, i.e., r is the maximum number of sniffers that are in the transmission range of an AP.*

Proof. Consider an arbitrary AP, v, and a sniffer $m \in M_v$. Our LP rounding guarantees that $x_{m,c_v} \leq r y_{m,c_v}$, where $r = \max_{v \in V} |M_v|$. This can be shown by considering the following two cases. When $y_{m,c_v} = \max_{m \in M_v} y_{m,c_v}$, by our LP rounding, $x_{m,c_v} = 1$, and we have $x_{m,c_v} \leq r y_{m,c_v}$ (since $y_{m,c_v} \geq 1/r$). When $y_{m,c_v} \neq \max_{m \in M_v} y_{m,c_v}$, by our LP rounding, $x_{m,c_v} = 0 \leq r y_{m,c_v}$. Since the above AP, v, is chosen arbitrarily, we have

$$\sum_{c \in C} x_{m,c} \leq r \sum_{c \in C} y_{m,c}, \forall m \in M.$$

Let n_m^* represent the optimal solution to the min-max sniffer channel selection problem. We have

$$\max_{m \in M} \sum_{c \in C} x_{m,c} \leq r(\max_{m \in M} \sum_{c \in C} y_{m,c}) \leq r n_m^*. \tag{4}$$

The second inequality above is because the LP provides a lower bound to the original problem. From (4), LP-min-max is an $O(r)$-approximation algorithm for the min-max sniffer channel selection problem. Similarly, let n_s^* represent the optimal solution to the min-sum problem. We have

$$\sum_{m \in M} \sum_{c \in C} x_{m,c} \leq r(\sum_{m \in M} \sum_{c \in C} y_{m,c}) \leq r n_s^*. \tag{5}$$

Hence LP-min-sum is an $O(r)$-approximation algorithm for the min-sum problem. □

2.4.3 Greedy-min-max

The main idea of Greedy-min-max is as follows. Initially, a sniffer, m, is assigned to monitor an AP, v, as long as m is in the transmission range of v. The algorithm then

Algorithm 2 Greedy-min-max

$\varphi(v) = \{m \mid m \in M_v\}, \forall v \in V$
$C_\varphi(m) = \emptyset$, $V_{m,c} = \emptyset$, $\forall m \in M, c \in C$
for all $v \in V$ **do**
  for all $m \in M$ **do**
    if $m \in M_v$ **then**
      $C_\varphi(m) = C_\varphi(m) \cup \{c_v\}$
      $V_{m,c_v} = V_{m,c_v} \cup \{v\}$
    end if
  end for
end for
$M_\varphi = M$
repeat
  $M' = \emptyset$
  for all $m \in M$ **do**
    if $\exists c \in C_\varphi(m)$ s.t. $\forall v \in V_{m,c}, |\varphi(v)| \geq 2$ **then**
      $M' = M' \cup m$
    end if
  end for
  if $M' \neq \emptyset$ **then**
    Suppose $m \in M'$ monitors the largest number of channels
    $C'_\varphi(m) = \{c \mid c \in C_\varphi(m), |\varphi(v)| \geq 2, \forall v \in V_{m,c}\}$
    Pick $c \in C'_\varphi(m)$ that has the smallest $|V_{m,c}|$
    $C_\varphi(m) = C_\varphi(m) \setminus \{c\}$
    if $C_\varphi(m) = \emptyset$ **then**
      $M_\varphi = M_\varphi \setminus \{m\}$
    end if
    $\varphi(v) = \varphi(v) \setminus \{m\}, \forall v \in V_{m,c}$
    $V_{m,c} = \emptyset$
  end if
until M' is empty
Return$(\varphi, C_\varphi, M_\varphi)$

runs in iterations. In each iteration, it finds the sniffer with the maximum number of channels and removes one channel from this sniffer when feasible (i.e., while still satisfying the monitoring constraints) since our goal is minimize the maximum number of channels that a sniffer uses. The iteration stops when none of the sniffers can remove any channel.

Algorithm 2 summarizes this algorithm. Line 1 initializes $\varphi(v)$ to be the set of sniffers that are in the transmission range of v, $\forall v \in V$. Let $V_{m,c}$ denote the set of APs that sniffer m monitors on channel c. Lines 2-10 initialize $C_{\varphi}(m)$ and $V_{m,c}$, $\forall m \in M, c \in C$. Line 11 initializes M_{φ}, the set of sniffers that are being used, to be the entire set of sniffers. In each iteration (lines 12-30), let M' record the set of sniffers that can remove at least one channel. It then picks a sniffer, m, that monitors the maximum number of channels from M'. Afterwards, it finds a channel, c, that can be removed and removes it from $C_{\varphi}(m)$. If multiple such channels exist, it chooses to remove the channel with the smallest number of APs (so that removing such a channel may affect the least number of APs). If after removing the channel, $C_{\varphi}(m)$ becomes an empty set, then we remove the sniffer m from M_{φ} (lines 24 to 26). Last, line 27 removes m from the assignment of all the APs in $V_{m,c}$ (since m does not monitor channel c any more), and line 28 sets $V_{m,c}$ to an empty set.

We now briefly describe the complexity of Greedy-min-max. The running time of lines 3-10 is $O(|V||M|)$. In each iteration of loop between lines 12-30, the algorithm picks one channel. So, in the worst case, the running time of this loop is $O(|M||C|)$. The sub-loop between line 14-18 can be finished within $O(|M||V||C|)$. We can neglect the running time of line 19-29. In summary, the running time of Greedy-min-max is $O(|M|^2|C|^2|V|)$.

We now illustrate Greedy-min-max using the example in Fig. 1(a). Initially, we assign each sniffer to monitor two channels at the beginning. In the iteration, we first set $M' = \{m_1, m_2, m_3\}$, and choose m_1, and then remove channel 1 from $C_\varphi(m_1)$. Set $\varphi(v_1) = \{m_2\}$, $\varphi(v_2) = \{m_2, m_3\}$, and $\varphi(v_3) = \{m_1, m_3\}$. In the second iteration, we choose m_2, and remove channel 2 from $C_\varphi(m_2)$. Then set $\varphi(v_2) = \{m_3\}$, $\varphi(v_3) = \{m_1, m_3\}$. In the last iteration, we choose m_3, and remove channel 3 from $C_\varphi(m_3)$. Therefore, $\varphi(v_3) = \{m_1\}$. In summary, the Greedy-min-max solution is $C_\varphi(m_1) = \{c_3\}$, $C_\varphi(m_2) = \{c_1\}$, and $C_\varphi(m_3) = \{c_2\}$. The assignment results are $\varphi(v_1) = \{m_2\}$, $\varphi(v_2) = \{m_3\}$, and $\varphi(v_3) = \{m_1\}$. This is also an optimal assignment, while it differs from the optimal solution from IP-min-max.

2.4.4 Remove redundant Sniffers

None of the above three algorithms explicitly minimizes the number of sniffers that are being used. As a result, the solutions may contain a large number of redundant sniffers. We next propose two algorithms to remove redundant sniffers. One is an optimal algorithm, based on IP, and the other is a heuristic algorithm that has much shorter running time.

The main idea of the IP-based algorithm is as follows. Given a sniffer channel selection solution $\varphi(\cdot)$, it exhaustively searches among the sniffers that are being used, and finds the maximum number of sniffers that can be removed while still maintaining that all the APs are being monitored by at least one sniffer. Specifically, the IP

formulation is

$$\text{minimize}: \quad z \tag{6}$$

$$\text{subject to}: \quad \sum_{m \in \varphi(v)} z_{m,c_v} \geq 1, \forall v \in V \tag{7}$$

$$\sum_{m \in M_\varphi} z_{m,c_m} \leq z, \forall c_m \in C_{\varphi(m)}, m \in M_\varphi \tag{8}$$

$$z_{m,c} \in \{0,1\}, \forall m \in M, \forall c \in C \tag{9}$$

$$z_{m,c} = 0, \forall m \notin M_\varphi, \text{ or } c \notin C_\varphi(m) \tag{10}$$

In this formulation, we define an integer variable z that represents the number of sniffers that are being used, and a set of binary variables $z_{m,c}$, where $z_{m,c} = 1$ indicates that sniffer m listens channel c, and $z_{m,c} = 0$ indicates otherwise, $m \in M$, $c \in C$. The objective function is minimizing z. The value of $z_{m,c}$ depends on the channel assignment solution $\varphi(\cdot)$: it is zero if sniffer m is not used or channel c is not in $C_\varphi(m)$; otherwise, $z_{m,c}$ can be either 0 or 1 (see constrains 9 and 10). The constraint (7) requires the solution provided by $z_{m,c}$ covers all of the APs. Since a sniffer is used if it monitors at least one channel, the sum of $z_{m,c}$ for $m \in M_\varphi, c \in C_{\varphi(m)}$ is no less than the number of sniffers that is used. The set of constraints in (8) lists all possible such summations (there are $\Pi_{m \in M_\varphi} |C_\varphi(m)|$ such summations).

We now apply the above IP-based algorithm to remove redundant sniffers in the solutions provided by IP-min-max, LP-min-max and Greedy-min-max for the example in Fig. 1(a). For the solution from IP-min-max, the constraint (8) is $z_{m_1,c_1} + z_{m_2,c_2} + z_{m_3,c_3} \leq z$; for the solution from LP-min-max, the constraint (8) is $z_{m_2,c_1} + z_{m_3,c_3} \leq z$ and $z_{m_2,c_2} + z_{m_3,c_3} \leq z$; and for the solution from Greedy-min-max, the constraint (8) is $z_{m_1,c_3} + z_{m_2,c_1} + z_{m_3,c_2} \leq z$. For all three solutions, solving the IP formulation (6)-(10), we find no sniffer can be removed.

The IP Algorithm needs exponential running time, and hence is not applicable to large-scale problems. We next propose a greedy algorithm, Algorithm 3, that has a polynomial running time to remove redundant sniffers. In Algorithm 3, $V_{m,c}$ denotes the set of APs that are monitored by sniffer m on channel c. Lines 1-6 initialize $V_{m,c}$ based on the sniffer channel assignment. Lines 7-14 consider all the sniffers that have been used, and for each sniffer, it checks all the channels that have been selected for the sniffer, and removes unnecessary channels (a channel is not necessary if all the APs are still monitored by at least one sniffer after removing this channel). At the end, lines 15-19 remove all the unnecessary sniffers (i.e., sniffers that do not monitor any channel) from M_φ.

Algorithm 3 Remove redundant sniffers (using a greedy heuristic)

1: $V_{m,c} = \emptyset$, $\forall m \in M_\varphi, \forall c \in C$
2: **for all** $v \in V$ **do**
3: **for all** $m \in \varphi(v)$ **do**
4: $V_{m,c_v} = V_{m,c_v} \cup \{v\}$
5: **end for**
6: **end for**
7: **for all** $m \in M_\varphi$ **do**
8: **for all** $c \in C_\varphi(m)$ **do**
9: **if** $\forall v \in V_{m,c}$, $\exists m' \neq m$ s.t. $v \in V_{m',c}$ **then**
10: $C_\varphi(m) = C_\varphi(m) \setminus \{c\}$
11: $\varphi(v) = \varphi(v) \setminus \{m\}, \forall v \in V_{m,c}$
12: **end if**
13: **end for**
14: **end for**
15: **for all** $m \in M$ **do**
16: **if** $C_\varphi(m) = \emptyset$ **then**
17: $M_\varphi = M_\varphi \setminus \{m\}$
18: **end if**
19: **end for**

The running time of Algorithm 3 is $\sum_{v \in V} |\varphi(v)| + \sum_{m \in M_\varphi} |C_\varphi(m)|$. It may not provide optimal solutions. We compare the performance of the two algorithms to remove redundant sniffers in Section 2.6.

2.5 Algorithms for min-sum sniffer channel selection

It is easy to see that the min-sum sniffer channel selection problem is NP-hard. This is because when there is a single channel, it is equivalent to the minimum set cover problem, which is NP-hard. We also develop three algorithms to solve it, based on IP, LP-relaxation, and a greedy heuristic, referred to as *IP-min-sum*, *LP-min-sum*, and *Greedy-min-sum*, respectively. After running each algorithm, we can again use the algorithms in Section 2.4.4 to remove redundant sniffers. We next describe the three algorithms in detail.

IP-min-sum differs from IP-min-max in that it first solves an IP problem with the objective function

$$\text{minimize} : \sum_{m \in M} \sum_{c \in C} x_{m,c} \tag{11}$$

instead of (1) as in IP-min-max. LP-min-sum differs from LP-min-max in that it first solves an LP-relaxation problem for the min-sum problem instead of the min-max problem. LP-min-sum has the same complexity as LP-min-max. We have a similar approximate ratio result for LP-min-sum, as stated in the following theorem. The proof is found in Section 2.4.1.

Theorem 2. *LP-min-sum is an $O(r)$-approximation algorithm for the min-sum sniffer channel selection problem, where $r = \max_{v \in V} |M_v|$, i.e., r is the maximum number of sniffers that are in the transmission range of an AP.*

Algorithm 4 Greedy-min-sum

1: $\varphi(v) = \emptyset, \forall v \in V$, $C_\varphi(m) = \emptyset, \forall m \in M$, $M_\varphi = \emptyset$
2: $V_{m,c} = \emptyset$, $\forall m \in M, c \in C$
3: **for all** $v \in V$ **do**
4: **for all** $m \in M$ **do**
5: **if** $m \in M_v$ **then**
6: $V_{m,c_v} = V_{m,c_v} \cup \{v\}$
7: **end if**
8: **end for**
9: **end for**
10: $V' = V$
11: **repeat**
12: pick m, c such that $|V_{m,c}| = \max_{m' \in M, c' \in C} |V_{m',c'}|$
13: $\varphi(v) = \varphi(v) \cup \{m\}, \forall v \in V_{m,c}$
14: $C_\varphi(m) = C_\varphi(m) \cup \{c\}$
15: $M_\varphi = M_\varphi \cup \{m\}$
16: $V_{m',c} = V_{m',c} \setminus V_{m,c}, \forall m' \in M$
17: $V' = V' \setminus V_{m,c}$
18: **until** V' is empty
19: Return$(\varphi, C_\varphi, M_\varphi)$

Greedy-min-sum models the sniffer channel selection problem as a minimum set covering problem: we map each sniffer to $|C|$ *virtual sniffers*, each monitoring one channel in C, then the min-sum problem is equivalent to finding the minimum number of virtual sniffers so that all APs are monitored and the number of virtual sniffers (and hence the sum of the channels used by all the sniffers) is minimized. Many algorithms have been proposed for the minimum set covering problem. Greedy-min-sum follows a greedy algorithm for minimum set covering problem [38]. It runs in iterations. In each iteration, it picks a sniffer and channel pair that can monitor the maximum number of APs. The iteration continues until all the APs are monitored.

Algorithm 4 summarizes this algorithm (we used a similar algorithm for scheduling sniffers to detect rogue APs in [76]). Let $V_{m,c}$ denote the set of APs that sniffer m could monitor if it listened to channel c. Line 1 initializes $C_\varphi(m)$, M_φ and $\varphi(v)$ to

be empty sets, $\forall m \in M, v \in V$. Lines 2-9 initialize $V_{m,c}$, $\forall m \in M, c \in C$. Line 10 initializes, V', the set of APs that have not been monitored, to V. The algorithm then runs in iterations until V' is empty. Using a greedy strategy, line 12 chooses a monitor and channel pair, m and c, that covers the maximum number of APs, i.e., $|V_{m,c}| = \max_{m' \in M, c' \in C} |V_{m',c'}|$ (if multiple such sniffers exist, we choose the one that monitors the minimum number of channels, i.e., with the minimum $|C_\varphi(m)|$). After choosing the monitor and channel pair, m and c, line 13 assigns m to all the APs in $V_{m,c}$; line 14 adds channel, c, into $C_\varphi(m)$; and line 15 adds monitor m to M_φ. Afterwards, since the APs in $V_{m,c}$ have already been monitored, line 16 removes $V_{m,c}$ from $V_{m',c}, \forall m' \in M$, and line 17 removes $V_{m,c}$ from V'.

Following the results in [38], the approximation ratio of Greedy-min-sum is H_d for the min-sum problem, where $H_d = \sum_{i=1}^{d} 1/i$ is the d-th harmonic number, and d is the maximum number of APs that a sniffer can monitor in its neighborhood. The complexity of Greedy-min-sum is $O(|M|^2|C|^2)$: the dominant complexity is in the loop between line 11-18; inside the loop, each iteration chooses one monitor m and one channel c, leading to $O(|M||C|)$ iterations, and inside an iteration, line 12 has running time of $O(|M||C|)$, and hence the total running time is $O(|M|^2|C|^2)$.

Last, we describe the assignment results when using the three algorithms to solve the example in Fig. 1(b) (all three algorithms obtain the same solution for the example in Fig. 1(a), details omitted in the interest of space). When using IP-min-sum, we have $x_{m_1,c_1} = 0$, $x_{m_1,c_2} = 1$, $x_{m_2,c_1} = 1$, $x_{m_2,c_2} = 0$, $x_{m_3,c_1} = 0$, and $x_{m_3,c_2} = 0$, leading to an optimal solution of two. Specifically, the assignment is $\varphi(v_1) = \varphi(v_2) = \{m_2\}$, $\varphi(v_3) = \{m_1\}$, and hence $C_\varphi(m_1) = \{c_2\}$, $C_\varphi(m_2) = \{c_1\}$, and $C_\varphi(m_3) = \emptyset$. When using LP-min-sum, we have $y_{m_1,c_1} = 0.5$, $y_{m_2,c_1} = 0.5$, $y_{m_3,c_1} = 0.5$, $y_{m_3,c_2} = 0.5$, $y_{m_1,c_2} = 0.5$, $y_{m_2,c_2} = 0$. Therefore, when selecting sniffer to monitor channel 1, y_{m_1,c_1}, y_{m_2,c_1}, and

y_{m_3,c_1} are all 0.5[5] . Suppose we choose m_1, the solution is $\varphi(v_1) = \{m_1\}$, $\varphi(v_2) = \{m_3\}$, $\varphi(v_3) = \{m_1\}$, and hence $C_\varphi(m_1) = \{c_1, c_2\}$, $C_\varphi(m_3) = \{c_1\}$, and $C_\varphi(m_2) = \emptyset$, leading to a suboptimal solution of 3. The solution obtained by Greedy-min-sum is the same as that by IP-min-sum. We again apply the algorithms in Section 2.4.4 to remove redundant sniffers, and find no sniffer can be removed for the solutions provided by the three algorithms.

2.6 Performance evaluation

Our performance evaluation uses two empirical datasets. One corresponds to the campus WLAN network in Dartmouth College. The other is obtained using Placelab[6] from Seattle downtown area. Both datasets are obtained by wardriving. The APs are deployed in buildings, and can be densely deployed at certain locations. The Dartmouth dataset represents a well-managed wireless network, while the Seattle dataset is a completely unmanaged network. The Dartmouth dataset contains both AP location (2D coordinates) and channel information. More specifically, the APs in the dataset use both 802.11b/g and 802.11a, and operate on 12 orthogonal 2.4GHz/5GHz channels. In the following, we treat each AP as two duplicate APs, each working on one channel. The Seattle dataset only contains AP location information. We randomly assign each AP one channel from the 24 available channels for 802.11b/g and 802.11a. The transmission range of an AP in both datasets is set to 100 m.

We first evaluate the two algorithms that remove redundant sniffers (see Section 2.4.4). Since the IP-based algorithm cannot solve large-scale problems in a reasonable time, we use a small network to compare these two algorithms. Specifically, we

[5]We again break ties arbitrarily, as in the LP-min-max algorithm.
[6]http://www.placelab.org/database/

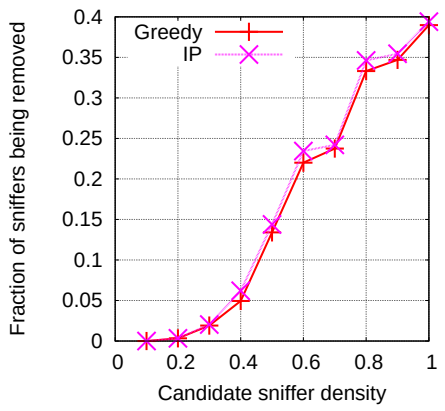


Figure 3: Performance of the two algorithms that remove redundant sniffers. The results are for an area with 25 APs in the Dartmouth dataset. The sniffer channel selection solution is obtained using IP-min-max.

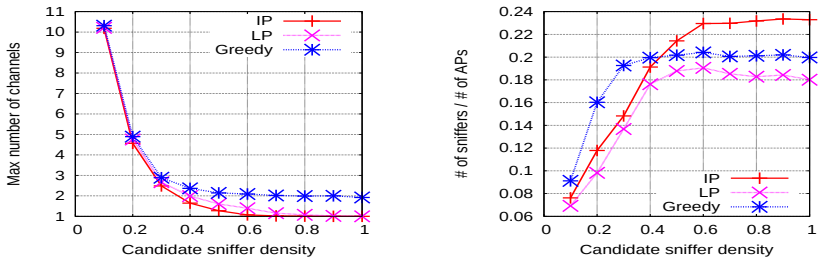
choose an area that contains 25 APs in the Dartmouth dataset. For systematic evaluation, we generate $1,000$ topologies by virtually placing candidate sniffers uniformly randomly into the area. The number of candidate sniffers is randomly chosen from 1 to the number of APs. For each topology, we obtain a pair (n_a, n_s), where n_a is the number of APs that can be monitored by at least one sniffer, and n_s is the number of sniffers that can monitor at least one AP (i.e., sniffers that are within the transmission range of at least one AP). Therefore n_a and n_s can be smaller than the number of APs and sniffers in the area, respectively. We refer to the ratio, n_s/n_a, as *candidate sniffer density*. For each topology, we run the IP-min-max algorithm to select channels for the sniffers, and then apply the two algorithms to remove redundant sniffers. Fig. 3 plots the fraction of sniffers that are removed versus candidate sniffer density, n_s/n_a. The results are aggregated over a bin size of 0.1, i.e., the result under $n_s/n_a = x$ is the average of all the topologies with $n_s/n_a \in (x - 0.1, x]$ (the confidence intervals are tight and hence omitted). The results from both the IP-based optimal algorithm and the greedy heuristic are plotted in the figure. We observe that the performance of the

greedy heuristic is close to that of the IP-based optimal algorithm. Considering the running time, the rest of the results in this section uses the greedy heuristic to remove redundant sniffers.

We now present the results when solving the min-max and min-sum problems in large-scale networks. For both the Dartmouth and Seattle datasets, we consider two 500 m $\times$ 500 m areas: one has approximately 400 APs, representing an area with densely deployed APs; the other has a much lower AP density (approximately 200 APs). To systematically evaluate the performance of our algorithms, for each area we consider, we again generate $1,000$ topologies by virtually placing candidate sniffers uniformly randomly into the area.

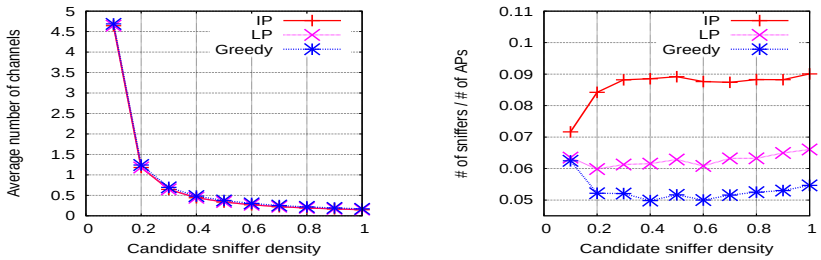
Our simulation runs on a Intel Xeon PC with four 3.0GHz processors. For each algorithm, the running time for the Dartmouth dataset is shorter than that for the Seattle dataset. This might be because the former uses 12 channels while the latter uses 24 channels. For both the min-max and min-sum problems, the LP-based algorithms are the fastest: it only takes a few minutes to finish solving all the $1,000$ topologies. The IP-based algorithms are the slowest: it can take up to 7 hours to solve the $1,000$ topologies. The running time of the greedy heuristics is in between (it takes tens of minutes to finish the $1,000$ topologies). In the following, we mainly present the results for the 400-AP area in the Dartmouth dataset; results under other settings (the other area in the Dartmouth dataset and the two areas in the Seattle dataset) are similar.

Fig. 4 plots the results when solving the min-max sniffer channel selection problem for the 400-AP area in the Dartmouth dataset. The results under all the three algorithms, IP-min-max, LP-min-max, and Greedy-min-max, are plotted in the figure. Fig. 4(a) plots the maximum number of channels that a sniffer monitors versus candidate sniffer density, n_s/n_a. The results are again aggregated over a bin size of 0.1



(a) (b)

Figure 4: The min-max sniffer channel selection problem: (a) maximum number of channels that a sniffer monitors, and (b) ratio of the number of sniffers that are being used over the number of APs. These results are for the 400-AP area in the Dartmouth dataset; IP, LP and Greedy are abbreviations for IP-min-max, LP-min-max, and Greedy-min-max, respectively.



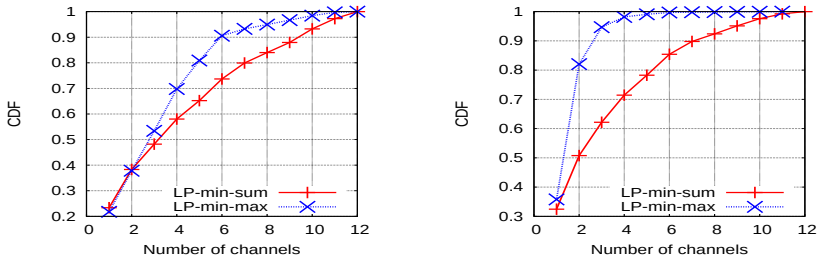
(a) (b)

Figure 5: The min-sum sniffer channel selection problem: (a) average number of channels that a sniffer monitors, and (b) ratio of the number of sniffers that are being used over the number of APs. These results are for the 400-AP area in the Dartmouth dataset; IP, LP and Greedy are abbreviations for IP-min-sum, LP-min-sum, and Greedy-min-sum, respectively.

(the confidence intervals are tight and hence omitted). We observe that for all three algorithms, as expected, the maximum number of channels used by the sniffers reduces as the candidate sniffer density increases. IP-min-max provides the optimal solution (in terms of the objective function). LP-min-max performs slightly worse than IP-min-max: the performances of these two algorithms are similar under both low and high sniffer densities; the difference is most noticeable for medium range of sniffer density (between 0.4 and 0.6). Both IP-min-max and LP-min-max outperform Greedy-min-max, particularly for large values of sniffer density. We also observe a diminishing gain

from increasing the density of sniffers: the maximum number of channels decreases dramatically first and then less dramatically afterwards. Fig. 4(b) plots the ratio of the number of sniffers that are being used over the number of APs. We observe that LP-min-max outperforms the other two algorithms: the fraction of the sniffers that are being used under LP-min-max is 10% to 20% lower than that under IP-min-max, and is 5% to 25% lower than that under Greedy-min-max. Taking account of both the objective function and the number of sniffers needed, LP-min-max is a preferred algorithm than the other two: the maximum number of channels under LP-min-max is only slightly larger under the optimal solution, while the number of sniffers needed by LP-min-max is significantly lower than the other two algorithms. Last, from Figures 4(a) and (b), we observe that for the min-max sniffer channel selection problem, a preferred candidate sniffer density is between 0.2 to 0.3, which leads to significant reduction in the maximum number of channels used by the sniffers compared to lower densities, while leads to moderate cost in deploying the air sniffing infrastructure: the number of sniffers that are being used is below 14% of the number of APs under LP-min-max, the preferred algorithm.

Fig. 5 plots the results for the min-sum sniffer channel selection problem when using the three algorithms, IP-min-sum, LP-min-sum, and Greedy-min-sum. Fig. 5(a) plots the average number of channels that a sniffer monitors versus candidate sniffer density, n_s/n_a. Each data point is the average calculated over all sniffers, excluding those that are not being used. We observe that all three algorithms lead to similar performance, both LP-min-sum and Greedy-min-sum provide solutions close to the optimal solution from IP-min-sum. Again, we observe a diminishing gain from increasing the density of sniffers. Fig. 5(b) plots the ratio of the number of sniffers that are being used over the number of APs. We observe Greedy-min-sum slightly outperforms LP-min-sum,



(a) Candidate sniffer density is 0.1. (b) Candidate sniffer density is 0.2.

Figure 6: Sniffer workload distribution when solving the min-max and min-sum sniffer channel selection problems for the 400-AP area in the Dartmouth dataset.

and significantly outperforms IP-min-sum (it requires 13% to 40% less sniffers than IP-min-sum). For all the setttings, the number of sniffers that are being used is much smaller than the number of APs (the former is 5% to 9% of the latter), indicating a moderate cost of deploying the air sniffing infrastructure.

Summarizing the above observations, we conclude that, considering running time, the objective function, and the fraction of sniffers that are used, the LP-based algorithm outperforms the IP-based and the greedy-heuristic based algorithms for both the min-max and min-sum problems. Considering both the running time and performance, LP-based algorithms maybe a preferable choice for large networks in practice.

Comparing Fig. 4(b) and Fig. 5(b), we see that, for the same candidate sniffer density, the min-sum problem requires much less sniffers than the min-max problem. For instance, when the sniffer density is 0.2, the ratio of the number of sniffers over the number of APs for the min-max problem is between 0.1 to 0.16, (see Fig. 4(b)), while the ratio for the min-sum problem is between 0.05 to 0.085 (see Fig. 5(b)). This is not very surprising: the min-max problem needs a larger number of sniffers since it requires the workloads of the sniffers to be balanced (to achieve the min-max goal). We next further compare the sniffer workload distribution for the min-max and min-sum problems. Fig. 6 plots the CDF (cumulative distribution function) of the sniffer

workloads (i.e., workload of a sniffer is the number of channels that is used by the sniffer), excluding those sniffers that are not used. Again, it is for the 400-AP area in the Dartmouth dataset. We only plot the results under the LP-based algorithms (i.e., LP-min-max and LP-min-sum) when the sniffer density is 0.1 and 0.2 (shown in Figures 6(a) and (b), respectively). From Fig. 6(a), we observe respectively 90% and 74% of the sniffers scan at most 6 channels in the min-max and min-sum problems. The difference is more dramatic in Fig. 6(b) where nearly 100% of the sniffers scan at most 4 channels in the min-max problem while the corresponding value is only around 70% in the min-sum problem.

2.7 Summary

In this paper, we studied sniffer channel selection for monitoring WLANs. In particular, we formulated min-max and min-sum sniffer channel selection problems, and proposed three algorithms, one based on IP, one based on LP-relaxation, and the third based on a greedy heuristic, to solve each problem. Through simulation, we demonstrated that for each problem, all the algorithms are effective in achieving their optimization goals, and overall, the LP-based algorithm outperform the other two algorithms.

Chapter 3

Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network

3.1 Introduction

Smart mobile handheld devices (MHDs) such as smartphones have been used for a wide range of applications including surfing web, checking email, watching video, accessing social networking services, and online games. Most MHDs are equipped with both cellular and WiFi interface cards. Whenever available, WiFi is still a preferred way for Internet connection due to its higher bandwidth, lower delay and lower energy consumption [28]. Although recently there is a flurry of studies on various aspects of smart MHDs (see Section 3.2), little is known about the network performance of smart MHDs in WiFi networks. Specifically, how is their network performance? What are the major factors that affect the network performance?

In this paper, we answer the above questions by measuring the performance of smart MHDs inside a university campus WiFi network. Specifically, our study is over a data set that is passively captured by a monitor placed at a gateway router in the

University of Connecticut (UConn). The data set is collected over three days (2.9TB of data), containing traffic from various wireless devices, including MHDs such as iPhones, iPod touches, iPads, Android phones, Windows phones, and Blackberry phones, and wireless non-handheld devices (NHDs) such as Windows laptops and MacBooks. To understand the performance of MHDs, we first separate the traffic of MHDs from that of NHDs. Our focus is on MHDs; we describe the results on NHDs when necessary for comparison.

Analyzing the data set, we find HTTP is the dominant traffic type, accounting for over 92% of the TCP flows. We therefore focus on the performance of HTTP flows in this paper. The behavior of HTTP is complicated: we find many HTTP flows contain multiple HTTP requests, and a significant portion of an HTTP flow is idling (with no data packets). Hence, the traditional throughput metric (i.e., the amount of data downloaded in a flow divided by the total duration of the flow) may introduce bias in measuring the performance of HTTP flows. We therefore define a metric, *per-flow servicing rate*, i.e., the amount of data downloaded corresponding to HTTP requests in a flow divided by the downloading duration of the flow, to quantify the performance of an HTTP flow (see Section 3.4). This metric is interesting in its own right: it represents the network performance of an HTTP flow, while excluding the effect of various delays (e.g., client processing delays and user pause times) that are irrelevant to network performance. Our main findings are as follows.

- We observe that the best performance is achieved for the MHD flows served by an Akamai server cluster that is located close to UConn. Furthermore, 16% of the MHD flows are served by this server cluster, accounting for 35% of the bytes. Overall, a large percentage (38.4%) of MHD flows are served by well provisioned Akamai and Google servers, account for 62% of the bytes. Interestingly, these

fractions are significantly larger than the corresponding values for NHDs, indicating that MHDs use Akamai and Google servers more heavily, which boosts their overall network performance. We also observe that Akamai and Google servers have adopted large initial congestion window, which contributes to the network performance of MHDs (particularly for short flows). On the other hand, we observe that most flows have low loss rates, indicating that loss rate is not a limiting factor in the campus WiFi network that we study.

- We find that MHDs tend to have longer local RTTs (i.e., delays within the university network) than NHDs. In addition, the number of concurrent flows has negative effect on performance, and the effect is more significant for MHDs than NHDs. These two differences between MHDs and NHDs might be caused by the inferior computation and I/O capabilities on MHDs. The effect of local RTT on network performance seems negligible due the fact that local RTT only takes a small portion of the RTT.

- We find that earlier versions of Android OS (before 4.X) cannot take advantage of the large initial congestion window adopted by many servers. While Android OS increases the receive window adaptively in every round trip time, it uses a very small initial receive window (much smaller than the initial congestion window adopted by Google and Akamai servers), which limits the performance of Andriod devices. In contrast, we observe that iOS uses a large static receive window, which fully exploits the benefit of large initial congestion window. On the other hand, most flows do not fully utilize the large receive window, potentially leading to unnecessary waste of resources on iOS devices.

- We find that some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. One example is the native YouTube application on iOS devices, which can use a large number of TCP flows to serve a single video. We suspect this is a design optimization for cellular networks, which is not suitable for WiFi networks.

Our findings highlight the impact of TCP parameters and application-level design choices on MHD network performance, providing valuable insights on content distribution, server provisioning, mobile system design, and application-level protocol design.

The rest of the paper is organized as follows. Section 3.2 describes related work. Section 3.3 describes data collection and classification. Section 3.4 introduces the performance metric. Section 3.5 describes our methodology. Section 3.6 presents network performance of MHDs, and explores the impact of network and application layer factors on the performance. Section 3.7 discusses what findings are specific to UConn WiFi network and what can be applied to other networks. Last, Section 3.8 concludes the paper and presents future research directions.

3.2 Related Work

Several recent studies characterize the usage and traffics of MHDs in 3G cellular networks, public WiFi or residential WiFi networks. Trestian et al. analyze a trace collected from the content billing system of a large 3G mobile service provider to understand the relationship among people, locations and interests in 3G mobile networks [68]. Maier et al. study packet traces collected from more than 20,000 residential DSL customers to characterize the traffic from MHDs in home WiFi networks [48]. Falaki et

al. employ passive sniffers on individual devices (Android and Windows Mobile smartphones) to record sent and received traffic, and provide a detailed look at smartphone traffic [33]. This study is limited to a small user population (43 users), and the traces were not separately analyzed for the different network interfaces (i.e., 3G and WiFi) being used, which have different properties. The study in [34] uses a larger population of 255 users, and characterizes intentional user activities and the impact of these activities on network and energy usage. Gember et al. compare the content and flow characteristics of MHDs and NHDs in campus WiFi networks [37]. We focus on network performance of MHDs in a campus WiFi network and the impact of various factors on the network performance of MHDs, which are not investigated in the above studies.

Huang et al. anatomize the performance of smartphone applications in 3G networks using their widely-deployed active measurement tool, 3GTest [41]. Tso et al. study the performance of mobile HSPA (a 3.5G cellular standard) networks in Hong Kong using extensive field tests [69]. Our study differs from them in that we study the network performance of MHDs in WiFi networks (instead of cellular networks). Furthermore, our study is based on large-scale traces collected passively from a university campus network, while the study in [41] adopts an active measurement tool and the study in [69] uses customized applications. The studies of [36, 57] report that iOS native YouTube player generates multiple TCP flows to stream a single video. We expand these studies by presenting the degree of inefficiencies caused by the large number of TCP flows and the main reason for the design choice in the player.

Several measurement studies are on university WiFi networks. However, MHDs have only been widely adopted recently, and none of those studies explores the network performance of MHDs as in our study. Our study builds upon the rich literature

on understanding the behavior of TCPs and content distribution in operational environments (e.g., [14, 32, 40, 44, 47, 50, 54, 67]), and our observations confirm some of the findings in those studies.

Last, several other aspects of MHDs have been studied recently, including battery use and charging behaviors [19, 56], energy consumption [18, 66], and performance enhancement techniques (e.g., [55, 80]). Our study differs in scope from them.

3.3 Data collection & Classification

We collect measurements at a monitoring point inside the University of Connecticut (UConn). The monitor is a commodity PC, equipped with a DAG card [9]. It is placed at a gateway router of UConn, capturing all incoming and outgoing packets through the router with accurate timestamps[1] . In particular, it captures up to 900 bytes of each packet, including application-level, TCP and IP headers. The campus network uses separate IP pools for Ethernet and Wireless LAN (WLAN). Since we are interested in the network performance of MHDs, which use the WLAN IP address pool, we use filters to capture only WLAN traffic.

We have collected two data sets. One data set is for three days, from 9am March 2 to 9am March 5, 2011. The other data set is for one day, from 9am April 23 to 9am April 24, 2012. Unless otherwise stated, we use the first data set (reasons for focusing on this data set and findings from the second data set are deferred to Section 3.6.9). In the following, we first provide a high-level view of the data, and then present methodology to separate MHD traffic from NHD traffic (the captured data set contains a mixture of traffic from both types of devices).

3.3.1 Data

Table 2 lists the number of packets captured during the three days. Overall, we collected over 5.8G packets (2.9 TB of data). Among them, 91.9% of the packets are carried by TCP. We only report the results obtained from the data collected on the first day; the statistical characteristics of the data on the other two days are similar[2] .

[1]The campus network balances loads among two gateway routers. The load balancing strategy is set up so that data packets and ACKs in a TCP flow are through the same router.

[2]The amount of traffic on Day 3 is less than that of the other two days since Day 3 is a Friday.

Table 2: The number of packets captured during the three days.

	Day 1	Day 2	Day 3	Overall
incoming pkts	1.2G	1.4G	0.8G	3.5G
outgoing pkts	0.8G	1.0G	0.5G	2.3G
total pkts	2.0G	2.4G	1.4G	5.8G
% TCP pkts	91.8%	92.7%	90.8%	91.9%

We say a TCP flow is *valid* if it finishes three-way handshaking and does not contain a RESET packet[3] . Among the valid TCP flows, we identify the applications using destination port numbers (a recent study shows that this simple approach provides accurate results [47]). Table 3 lists the most commonly used applications, and the percentages of their traffic over all TCP traffic in terms of flows, packets and bytes, where the number of bytes in a packet is obtained from the IP header. We observe a predominant percentage (92.3%) of the TCP flows are HTTP flows. This is consistent with measurement results in other campus networks [37] and home environments [47]. In the rest of the paper, we focus on the network performance of HTTP flows; the performance of other protocols is left as future work.

Table 3: Percentage of the traffic from commonly used applications (in terms of flows, packets and bytes) over all the TCP traffic (Day 1).

Application	flows	packets	bytes
HTTP (80)	92.3%	82%	86.7%
HTTPS (443)	4.3%	8%	5.4%
IMAPS (993)	0.2%	0.28%	0.13%

3.3.2 Data classification

We refer to an HTTP flow that contains packets coming from and/or destinating to an MHD as an MHD flow (similar definition for an NHD flow). Since our captured

[3] 11.9% of the flows cotain a RESET packet.

data contains a mixture of MHD and NHD flows, we need to separate these two types of flows. The first approach we use to differentiate MHD and NHD flows is based on the keywords in the user-agent field in HTTP headers (MHDs and NHDs use different keywords) [37, 48]. Once the type of a flow is identified using the user-agent field, it provides us information on the type of the associated device (i.e., whether it is an MHD or NHD), which can be used to further identify the types of other flows. Specifically, if we know that a flow, f, from a device (the device is represented using an IP address) is an MHD flow, then we know all the flows that are concurrent with f and have the same source IP address as f are MHD flows[4] . If the type of an HTTP flow is not identified using the above two approaches, we use the knowledge that an IP address pool is dedicated to Apple mobile devices at UConn[5] , and hence a flow using an IP address from this pool is an MHD flow. Using the above three approaches, we categorize 94.1% of the HTTP flows to be either MHD or NHD flows (91.9% of them are directly identified through the user-agent field, 6.7% are identified through the concurrent-flow approach, and the rest 1.4% are identified using the knowledge of the dedicated IP address pool). Specifically, we identify 0.7M MHD flows and 10.4M NHD flows, containing 38.8M and 821.1M packets, respectively. Further separation of the flows using other heuristics (e.g., TTL [48]) is left as future work.

We further identify the operating systems used by the MHDs and NHDs using the user-agent field. For MHDs, the dominant operating system is iOS, used in 96.2% of the MHD flows (iOS is used by iPhone, iPod touch, and iPad, with the percentages of flows as 65%, 26%, and 9%, respectively); and Android is the second most popular

[4]We can only classify the flows that are concurrent with f due to IP reassignment which can assign the same IP address to another device at another point of time.

[5]UConn network administrators set aside this dedicated pool to ease the management of IP addresses. Specifically, a device is assigned an IP address from this dedicated pool if its host name indicates that it is an Apple mobile device during the DHCP request phase.

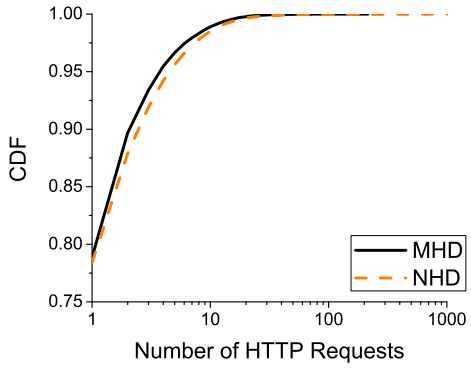


Figure 7: Distribution of the number of HTTP GET requests in an HTTP flow.

operating system, used in 3.2% of the MHD flows. For NHDs, the two dominant operating systems are OS X and Windows, taking 56.7% and 42.2% of the NHD flows, respectively; and Linux is at the third place, used by 1.1% of the NHD flows.

3.4 Performance Metric

As mentioned earlier, we mainly characterize the performance of HTTP flows since HTTP is the predominant traffic type. The behavior of HTTP is complicated. For instance, modern browsers often open multiple TCP connections when browsing a web page [32]. Furthermore, an HTTP flow can send and receive multiple HTTP requests/responses. In our passive measurements, without any knowledge of the accessed content, it is difficult to correlate multiple TCP connections. We therefore focus on per-flow network performance, specifically, the performance that is the result of the complex interactions of myriad network and application related factors. In the following, we first describe our measurement results on HTTP flows, and then define a performance metric that we will use in the rest of the paper.

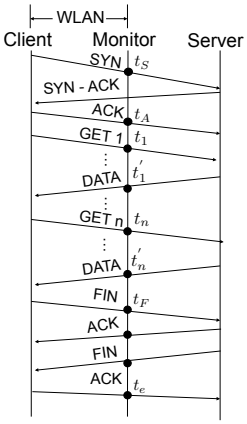
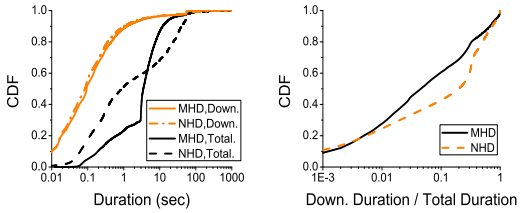


Figure 8: Illustration of a persistent HTTP flow.



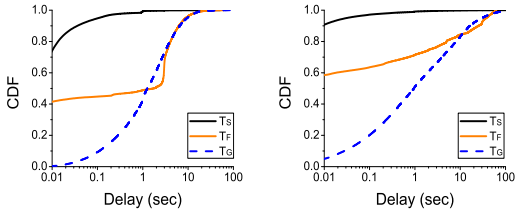
(a) (b)

Figure 9: (a) Total and downloading durations of MHD and NHD flows. (b) The ratio of downloading duration over total duration.

We find that over 99% of the HTTP flows use HTTP 1.1, where TCP connections are persistent, i.e., each TCP connection allows multiple GET and POST commands. On the other hand, we observe only 6.5% of the HTTP flows contain POST commands. This is not surprising since most of the users in the campus network are content consumers. In the following, we ignore the HTTP flows with POST commands. Fig. 7 plots the distribution of the number of HTTP GET requests within an HTTP flow. Observe that over 20% of the HTTP flows contain at least two requests. The number of requests in a flow can be as large as 100. In addition, NHD flows tend to contain a larger number of requests than MHD flows. This might be because regular web pages accessed by NHDs contain more objects than their mobile versions that are often accessed by MHDs [79].

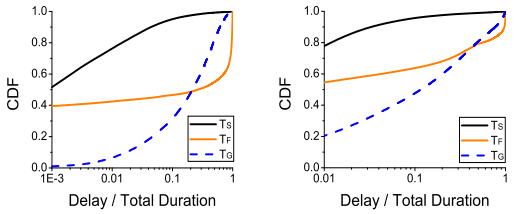
Fig. 8 illustrates a persistent HTTP flow with n requests (the requests are sequential since we do not observe any pipelined requests in our trace where a request is sent out without waiting for the response of the previous request[6]). The measurement point sits between the client and server, capturing the packets with accurate timestamps. In particular, let t_S denote the timestamp of the SYN packet, t_A denote the timestamp of the ACK packet in the three-way handshaking, t_i denote the timestamp of the ith HTTP request, t_i' denote the timestamp of the last data packet corresponding to the ith HTTP request, t_F denote the timestamp of the first FIN packet, and t_e denote the timestamp indicating the end of a TCP flow. Then the measured duration of the HTTP flow at the measurement point is $t_e - t_S$, while the duration for data downloading is $\sum_{i=1}^{n}(t_i^{'} - t_i)$. We refer to the former as *total duration* and the latter as *downloading duration*. Fig. 9(a) plots the distributions of the total and downloading durations for MHD and NHD flows. We observe that total durations can be significantly longer

[6]HTTP pipeline is not recommended and disabled in most browsers [10].



(a) MHD (b) NHD

Figure 10: Distribution of the various delays in an HTTP flow.



(a) MHD (b) NHD

Figure 11: Distribution of the ratios of the various delay over the total duration in an HTTP flow.

than downloading durations: the median downloading durations are 0.63s and 0.93s for MHD and NHD flows, respectively, while the median total durations are 5.06s and 15.45s for MHD and NHD flows, respectively. As shown in Fig. 9(b), the downloading duration is less than 10% of the total duration for respectively 60% and 42% of the MHD and NHD flows.

The difference between the downloading and total duration contains various delays inside an HTTP flow. Parts of the delays are for the three-way handshaking and TCP connection termination, and the rest of the delays are application idle/processing time. We divide the application idle/processing time into three parts. The first is the delay from finishing the three-way handshaking to the first GET request, defined as $T_S = t_1 - t_A$. The second is the sum of the delays from finishing one HTTP request to starting the next HTTP request, i.e., $T_G = \sum_{i=1}^{n-1}(t_{i+1} - t'_i)$, and the last is the delay from finishing downloading the last object to start to close the TCP connection, i.e.,

$T_F = t_F - t'_n$. Fig. 10 plots the distributions of the various delays. We observe that T_S is small for both MHD and NHD flows. Although a large fraction of the T_F values is small, it can be as large as several seconds for MHD flows and tens of seconds for NHD flows. The values of T_G are in a wide range from milliseconds to tens of seconds. Fig. 11 plots the CDF of the ratios of the various delay over the total duration.

In summary, the drastic difference between total duration and downloading duration highlights the importance of defining performance metric carefully: a traditional throughput metric, defined as the total amount of data downloaded divided by the total duration, can lead to biased results on the performance of HTTP flows. To exclude the effects of application-level delays on network performance, we define a performance metric, *per-flow servicing rate*, i.e., the total number of data bytes that are downloaded corresponding to HTTP requests divided by the downloading duration, to represent network performance. This metric represents the rate at which data are being fetched in an HTTP flow from a server to a client, while excluding the effect of various delays (e.g., client processing delays and user pause times) that are irrelevant to network performance.

3.5 Methodology

In the traces that we collected, MHDs are typically clients, requesting contents from servers outside UConn campus network. To understand the performance of MHD flows, we first group them according to their destinations. The rationale is that since the clients (i.e., MHDs) are at the same geographic location, the destinations (corresponding to the content servers) directly determine the network path. For the flows that are served by the same group of servers, we further divide the flows according to their lengths (in terms of number of packets inside the flow), since flow length affects

the impact of the various TCP parameters and network path conditions on servicing rate. Specifically, short flows may terminate before finishing the slow-start phase in TCP, while long flows are more likely to reach congestion avoidance phase, and hence achieve more steady throughput.

In the following, we first describe server and flow classification, and then describe how we measure the various TCP and application-level characteristics.

3.5.1 Server Classification

We use two steps to study each flow destination IP address. First, we use reverse DNS lookup to find its domain name, and then query the IP registration information to determine the organization that registered the IP address. For the IP addresses that do not have a published domain name, we use the registration information only.

Table 4: Top five domain names for MHD and NHD flows (percentage is in terms of flows).

MHD	NHD
Akamai.com (26.1%)	Akamai.com (20.0%)
Google.com (12.3%)	Google.com (8.9%)
Facebook.com (10.0%)	Facebook.com (8.3%)
Amazon.com (6.8%)	Yahoo.com (5.9%)
Apple.com (5.0%)	Amazon.com (5.3%)

Table 4 presents the top five domain names of MHD flows that are obtained using reverse DNS lookup. We see that the largest percentage of MHD flows are served by Akamai CDN. Indeed, many widely used applications on MHDs are served by Akamai. For instance, when an iOS device downloads an application from Apple App Store, the data are actually downloaded from Akamai. In addition, some popular destinations, e.g., "fbcdn.net", are served by Akamai (we find 12% of MHD flows are destined to "fbcdn.net"). Following Akamai.com, Google.com is the second most frequently

accessed domain name. This is not surprising: Google.com is one of the most frequently accessed web sites [5], and many MHDs have embedded Google applications (e.g., Google Map, Google Voice, Gmail). The third one is Facebook.com, which uses Akamai to serve many static contents, and use its own servers to serve many dynamic contents directly. The fourth one is Amazon.com, whose cloud services serve many popular applications running on iOS (e.g., Foursquare).

Since Akamai.com and Google.com are the top two domain names, we detail the network performance of the MHD flows that are served by Akamai and Google servers, as well as those served by the rest of the servers in Section 3.6. We also observe from Table 4 that 38.4% of MHD flows are served by Akamai CDN and Google servers. This percentage is significantly larger than the corresponding value for NHD flows (i.e., 28.9%). We believe the reason why MHD flows use Akamai and Google servers more heavily is that the destination hosts accessed by MHDs are less diverse than those accessed by NHDs [37]. Based on the host field in the HTTP request headers, we find around 6k distinct destination host domains from MHDs and 51k distinct destination host domains from NHDs. In addition, Table 5 lists the top 10 destination host domains accessed by MHD and NHD flows. We see for MHDs, accesses to the top ten host domains account for 39.2% of the MHD flows, while the percentage for NHDs (32%) is much lower. Last, we also observe from Table 5 that for MHDs, except for facebook.com and pandora.com, the other top eight host domains are all served by Akamai and Google, confirming the heavy usage of Akamai and Google servers by MHDs.

Table 5: Top ten destination host domains for MHD and NHD flows (percentage is in terms of flows).

MHD	NHD
fbcdn.net (12.0%)	fbcdn.net (12.9%)
facebook.com (10%)	facebook.com (6.5%)
apple.com (3.9%)	google.com (2.4%)
googlevideo.com (3.8%)	doubleclick.net (2.0%)
google.com (2.5%)	ytimg.com (1.7%)
admob.com (1.6%)	quantservec.om (1.6%)
doubleclick.net (1.5%)	yieldmanager.com (1.5%)
youtube.com (1.4%)	tumblr.com (1.2%)
google-analytics.com (1.3%)	twitter.com (1.1%)
pandora.com (1.2%)	yahoo.com (1.1%)

3.5.2 Flow Length Classification

For the flows that are served by the same server category, we divide the flows according to their lengths into three groups, denoted as G_1, G_2, and G_3. Specifically, G_1 contains short flows, with one to ten data packets, G_2 and G_3 contain longer flows, with at least 10 and 50 data packets, respectively. We make the above grouping since many web pages belong to G_1 [32], while long video streaming sessions may belong to G_3[7] . Table 6 lists the number of flows, packets and bytes (calculated from packet size in IP header) of the various groups. Observe that around 75% of the flows are short G_1 flows.

Table 6: Information of the various groups of MHD flows (for the data collected on Day 1).

	G_1	G_2	G_3
flows	0.5M	0.2M	0.03M
packets	2.9M	25.1M	19.0M
bytes	2.3GB	22.8GB	19.4GB

[7]We also consider another group, G_4, which contains flows with at least 100 packets. The results of G_4 are similar to those of G_3, and hence are not reported in this paper.

3.5.3 TCP flow characteristics

We now describe how we estimate the various TCP flow characteristics, including RTT, loss rate, local RTT and loss rate, server's initial congestion window, client's advertised receive window, and the maximum window size.

We use the techniques in [44] to obtain a sequence of RTT samples for a flow, and use the median to represent the flow's RTT. Packet losses are detected when observing triple duplicate ACKs, or a retransmission that is caused by timeout[8] . Local RTT represents the delay that a TCP flow experiences locally, inside a local-area network (i.e., the UConn campus network in our context). Similarly, local loss rate represents the loss rate that a TCP flow experiences inside a local-area network. In our study, both local RTT and local loss rate can be directly obtained from the measurements at the monitoring point since it is at the edge of the local-area network (see details in [22]).

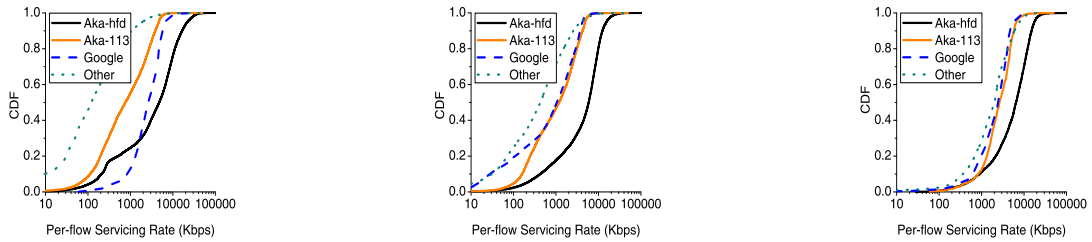
The server's initial congestion window size (icwnd) has a significant impact on the flow downloading rate. It determines the amount of data that the server can send in the first window. If the icwnd is too small, the sender needs to pause and waits for ACKs before transmitting again; on the other hand, if it is too large, the sender pushes too much data into the network, and congestion could happen [32]. We infer the icwnd based on the packet arrival times at the monitoring point. Specifically, we obtain an estimate of the icwnd as n when observing the first window of n packets arriving close in time at the the monitoring point. The first window is dynamically estimated based on RTTs using the technique in [44]. This approach will underestimate the icwnd if the application does not have sufficient amount of data to send in the first window.

[8]These are inferred losses at the TCP level. The loss rate thus estimated is an overestimate since the inferred losses may be due to long delays, not actual losses.

To avoid potential underestimation, we only apply the approach to HTTP flows that contain at least one GET request and the data corresponding to the first GET request contain at least 20 packets. The constraint of at least 20 data packets (corresponding to at least 20KB when each packet is at least 1000 bytes) is based on the literature that servers can choose a large icwnd, e.g., between 10KB to 20KB [32, 54]. To obtain a server's icwnd, we infer a series of icwnd values from all of the flows that are served by this server and satisfy the requirements stated earlier, and then obtain the average, median, and maximum from these inferred values (our measurements indicate that a server's icwnd can vary among the flows). In Section 3.6.4, we only report the results from servers that have at least 10 estimates.

Client's advertised receive window, constantly reported from the client to the server, states the size of the available receive buffer, i.e., the maximum size of data that a sender transmits to the receiver before the data is acknowledged by the receiver. The receive window field in the TCP header is 16 bits, limiting the maximum size of receive window to 64KB. However, if both TCP server and client support window scaling option, they can agree upon a window scaling factor, which is the multiplier to the 16-bit receive window field. For example, if window scaling factor is 4, the maximum receive window that a client can specify is 256KB.

At each time point, the server's available window is the minimum of the client's advertised receive window and server congestion window. We also measure the maximum window of each flow during its lifetime. The maximum window has impact on the network performance as well: it determines the maximum amount of data that the sender can transmit in one RTT, which affects the maximum downloading rate.



(a) G_1 (b) G_2 (c) G_3

Figure 12: Per-flow servicing rate of MHD flows served by the four server clusters.

3.5.4 Application layer characteristics

We consider the following three application-level characteristics. The first is the design of application-level protocols. In particular, we consider the protocol for YouTube videos. This is motivated by the popularity of this application and the large amount of video data in our trace (38% of the data are video contents). Secondly, we examine how servers respond to requests for different types of contents (e.g., video, image and texts). For this purpose, we define *application response time* as the delay from the first GET message to the first responding data packet for an HTTP request (the delay is measured at the monitoring point). Last, we quantify the relationship between the number of concurrent TCP flows and per-flow servicing rate. The rationale behind this is that the number of concurrent TCP flows might be an indicator of the amount of CPU and I/O activities on an MHD device. We calculate the amount of concurrent TCP flows for a flow as follows. Consider a flow f. Since the number of its concurrent flows can vary over time, we divide time into 0.3s bins, and count the number of concurrent flows in each bin, and then obtain the average number of concurrent flows during f's life-time as its number of concurrent flows. We only consider concurrent flows for long flows, specifically, the flows in G_3.

3.6 Network Performance of MHDs and Related Factors

In this section, we first present network performance of MHDs and then investigate how network and application layer factors affect the performance. Unless otherwise stated, we mainly report the results of iOS devices (i.e., iPhone, iPod touch, iPad) that account for 96.2% of the MHD flows.

As shown in Table 5, a large percentage (38.4%) of MHD flows are served by Akamai and Google servers. Using a commercial database [42], we identify that the Akamai servers are located in four main clusters, respectively 40km, 113km, 517km, and 966km away from UConn. Furthermore, consistent with the Akamai DNS design policy that gets servers close to end users [40], we find 90% of the MHD Akamai flows are served by the first two close-by server clusters. IP registration information reveals that all Akamai servers in the 40km range belong to the University of Hartford (only one hop away from UConn network), while it does not reveal any detailed information for the servers in the other distance ranges. We therefore refer to the first two server clusters as Akamai-Hartford and Akamai-113km, respectively. In the rest of the paper, we classify the servers into four clusters: Akamai-Hartford, Akamai-113km, Google[9] , and other servers.

Figures 12(a), (b), and (c) plot per-flow servicing rate distributions of G_1, G_2 and G_3 flows, respectively. The results for all four server clusters are plotted in the figure. For the same server cluster, we observe larger per-flow servicing rate for longer flows because longer flows allow the congestion window to ramp up. Overall, Akamai-Hartford servers provide the best performance, with median servicing rates of 4.7Mbps,

[9]Google does not publish the location information of its data centers. Therefore we cannot simply use the database [42] to determine the geographic locations of Google servers. Hence we present the aggregate results of all Google servers. It is possible to determine the geographic location of the servers using RTT [67]. Dividing the servers into clusters according to their geographic locations and investigating their respective performance are left as future work.

5.9Mbps, and 6.5Mbps for G_1, G_2, and G_3 flows, respectively. For G_2 and G_3 flows, we observe a clear stochastic order: the performance achieved by Akamai-Hartford servers is the best, followed by Akamai-113km, Google, and the other servers. For G_1 flows, the performance achieved by Google servers is superior to that of Akamai-113km servers, and is superior to that of Akamai-Hartford servers at low percentiles, a point that we will return later.

Existing studies (e.g., [57,59]) have shown that video servers may control the sending rate of video flows, which can affect per-flow servicing rate of those flows. In the traces that we collected, we confirm that YouTube flash videos to NHDs are indeed rate controlled. On the other hand, videos to NHDs are predominantly mp4 videos, which are not rate controlled. Therefore, per-flow servicing rates of MHD flows presented in Fig. 12 are not affected by rate control mechanisms. We next investigate the impact of various network and application-layer factors on MHD network performance.

3.6.1 RTT

We observe that RTTs to Akamai-Hartford servers tend to be the lowest, followed by Akamai-113km, Google, and the other servers. The main reason for the lowest RTTs to Akamai-Hartford servers is the close geographic distance and good network provisioning (both the University of Hartford where the servers are deployed and UConn belong to Connecticut education network). Fig. 13 plots the RTT distributions of G_2 MHD flows served by the four server clusters (results for G_1 and G_3 flows are similar). The median RTT of the flows served by Akamai-Hartford servers is 8.5ms, while for Akamai-113km, Google, and other servers, the median RTTs are respectively 2.3, 4.3, and 10.3 times larger. For the four server clusters, their relative order in terms of RTT is consistent with their relative performance as shown in Fig. 12. That is, a server

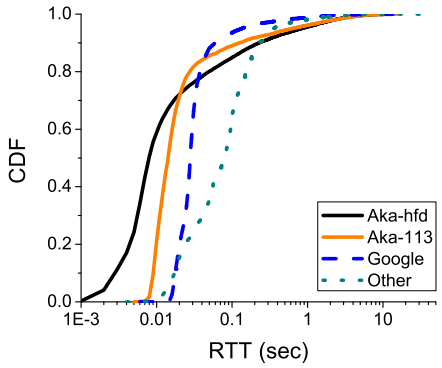


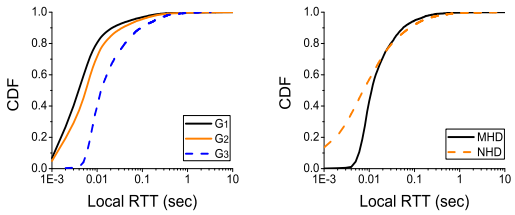
Figure 13: RTT distributions of G_2 flows served by the four server clusters.

cluster with lower RTTs tends to provide better performance. This is not surprising since RTT is a major factor that affects network performance.

What is more interesting is perhaps the large percentage of data that is served by the first three server clusters. Specifically, 58% of the MHD data bytes are served by the first three server clusters, much larger than the corresponding value (41%) for NHDs. Furthermore, 16% of the MHDs flows (accounting for 35% of the data) are served by Akamai-Hartford servers, the server cluster that provides the best performance. As described earlier, we believe that these large percentages originate from the fact that the most popular services accessed by MHDs are provided by major companies such as Facebook, Google, and Apple that use CDN services heavily. The large percentage of data served by Akamai and Google servers boost the overall network performance of MHDs.

3.6.2 Local RTT

We observe that long MHD flows tend to experience larger local RTTs than short MHD flows. As an example, Fig. 14(a) plots local RTT distributions of G_1, G_2, and



(a) (b)

Figure 14: (a) Local RTT for the MHD flows served by Akamai-Hartford servers. (b) Local RTT of G_3 MHD and NHD flows served by Akamai-Hartford servers.

G_3 MHD flows served by Akamai-Hartford server cluster (results for the other three server clusters are similar). We see that local RTT of G_1 flows is smaller than that of G_2 flows, which is in turn smaller than that of G_3 flows. In addition, we observe that local RTT of MHDs tends to be larger than that of NHDs. One example is shown in Fig. 14(b), which plots local RTT distributions of G_3 MHD and NHD flows served by Akamai-Hartford servers. Using *t-test* [43], we confirm that the above two observations indeed hold statistically. Specifically, for the MHD flows served by the same server cluster, the average local RTT of G_i flows is statistically smaller than that of G_j flows, $i < j$; and for G_i flows, the average local RTT of MHD flows is statistically larger than that of NHD flows, $i = 1, 2, 3$. The reason might be limited computation and I/O capabilities on MHDs, which lead to longer processing time for longer flows, as well as longer processing time than that on NHDs (a more in-depth study using active measurements is left as future work). On the other hand, except for Akamai-Hartford servers, local RTT is only a small portion of RTT (the ratio of local RTT over RTT is below 0.2 for 10% to 56% of the flows). Therefore, the impact of local RTT on per-flow servicing rate is negligible. We also observe a small fraction of local RTTs that are longer than 100ms, a point we will return to in Section 3.6.3.

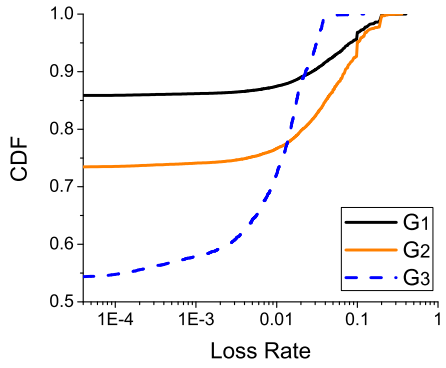


Figure 15: Loss rate distribution for the MHD flows served by Akamai-Hartford servers.

3.6.3 Loss Rate

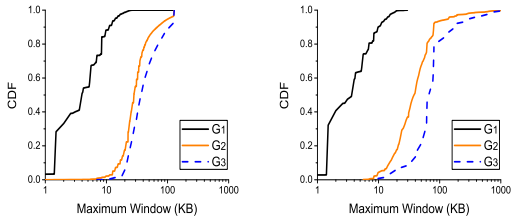
We find that most flows have very low loss rates, indicating that loss rate is not a limiting factor in the campus WiFi network that we study. An example is shown in Fig. 15, which plots loss rate distribution of the MHD flows served by Akamai-Hartford servers (results for the other three server clusters have similar trend). We observe that, for G_1, G_2 and G_3 flows, respectively 86%, 74% and 54% of the flows have zero loss. Furthermore, 90% of G_3 flows have loss rates below 0.02. The loss rate of a short flow can be large, which is however due to the artifact of small flow size (i.e., even a small number of losses can lead to high loss rate). We also find that all of the losses occur outside UConn (i.e., local loss rate is zero). This, however, does not mean that there is no loss at the MAC layer. We observe that a small percentage (around 3%) of local RTTs are longer than 100ms (see Fig. 14(a)). These long local RTTs might be caused by MAC-layer retransmissions and back-off times. Our measurements captured at the monitoring point does not contain MAC layer information; validating this conjecture through measurements captured at MAC layer is left as future work.

3.6.4 Initial Congestion Window

We find that Akamai and Google servers have adopted large initial congestion window. Specifically, the median initial congestion window of Akamai-Hartford servers is between 5 and 6KB, and the maximum initial congestion window is between 10 and 12KB. For Akamai-113km servers, 79% and 19% of the servers have median initial congestion window of 8KB and 5.5KB, respectively; the maximum initial congestion window is as large as 15KB. Google's adoption of large initial congestion window is even more aggressive: 98% of the Google servers use a median initial congestion window of 14KB (consistent with the advocated value of 15KB [32]), and the largest initial congestion window can be as large as 25KB. The rest of the servers (i.e., those other than Akamai and Google servers) have not adopted large initial congestion window as aggressively. Specifically, 61% of them use initial congestion window smaller than 4KB.

Large initial congestion window is particularly beneficial to short flows that can be completed in one RTT when initial congestion window is large. The more aggressive initial congestion window adopted by Google servers leads to its superior performance in serving G_1 flows: we observe from Fig. 12(a) better performance from Google servers than Akamai-113km servers at low percentiles despite that RTT to Google servers tends to be larger than that to Akamai-113km (RTT distributions for G_1 flows are similar to those for G_2 flows, which is plotted in Fig. 13). Furthermore, a smaller fraction of G_1 flows served by Google servers has very low servicing rate, as shown in Fig. 12(a). Even for G_2 flows, we observe similar servicing rates between flows served by Akamai-113km and Google servers (Fig. 12(b)) despite of the more dramatic differences in RTT (Fig. 13).

On the other hand, adopting larger initial congestion window can lead to congestion in the network. The study in [32] shows that this is only true for slow network



(a) iOS (b) Android

Figure 16: Distribution of maximum window size for iOS and Android flows served by Google servers.

connections. From our results, we do not observe significantly different loss rates from the server clusters that adopt different initial congestion windows.

3.6.5 Advertised Receive Window

We find that both iOS and Andriod operating systems support window scaling. When connecting to servers that support window scaling (e.g., we find all Akamai and Google servers support window scaling), the receive window advertised by iOS devices is 128KB[10] , implying that the maximum window of a flow is 128KB. In contrast to the static large receive window advertised by iOS devices, Andriod devices dynamically adjust receive window every round trip time, starting from a fairly small size of 5KB. Furthermore, since the receive window can grow dynamically, the maximum window size can be larger than 128KB.

The different design choices adopted by iOS and Andriod devices have the following implications. First, we find the large receive window of iOS devices cannot be fully utilized by most flows. To illustrate this, we plot the distribution of maximum window size of iOS flows that are served by Google servers, as shown in Fig. 16(a). The reason for choosing Google server cluster is that it serves the highest percentage of video flows

[10]The windows scaling factor is 4, leading to receive window of 256KB. However, we observe that the receive window is reduced to 128KB at the end of three-way handshake.

among all the four server clusters (the percentage is 30.5%), and video flows tend to be longer than other types of flows, and hence are more likely to reach large window size. From Fig. 16(a), we see that only 8% of G_3 flows reach the maximum window limit of 128KB, and the percentages for G_1 and G_2 flows are even lower, indicating that the large receive window of 128KB could potentially cause unnecessary waste of resources on iOS devices (whether it wastes resources or not depends on the kernel memory allocation mechanism: If the iOS kernel preallocates the memory for the socket, this causes waste of resources; otherwise, it does not waste resources).

Secondly, the very small receive window adopted by Andriod devices, while conserving resources, making Andriod devices unable to take advantage of large initial congestion window adopted by many servers (recall TCP congestion window is the minimum of the sender and receiver window), and hence can lead to inferior performance, particularly for short flows. As we shall see in Section 3.6.9, this is indeed the case (Section 3.6.9 reports findings from the data set collected in 2012, which contains more Andriod traffic and hence allows us to draw more convincing statistical conclusions regarding Andriod traffic). The adverse effect of small receive window may be even more dramatic in cellular networks where the round trip time can be significantly larger than that in WiFi networks.

Thirdly, since the receive window of Andriod devices is adjusted dynamically, it can grow to large values. Fig. 16(b) plots the distribution of maximum window size for Android flows that are served by Google servers. Indeed, we observe 9% of G_3 flows reach maximum window sizes larger than 128KB, while the maximum window size is bounded below 128KB for iOS flows. We suspect that since the receive window of Andriod devices can grow to large values, Android devices can achieve better performance for very long flows. Our dataset, however, does not contain sufficient number of very

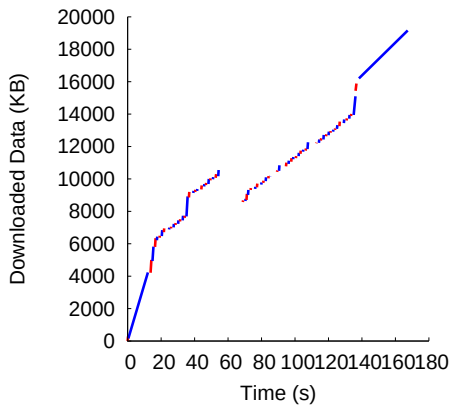


Figure 17: A series of 75 TCP flows is used to download one YouTube video when using the native YouTube application on an iOS device. The figure shows the starting and ending times of each TCP flow with the requested range. For better clarity, we use different colors (red and blue) to represent two adjacent TCP flows.

long flows from Android devices (even in the data set that was collected in April 2012, which contains much more traffic from Android devices) to statistically verify the above conjecture.

Summarizing the above, we believe dynamic receive window adopted by Andriod devices is a suitable choice for resource limited MHDs. On the other hand, using very small initial receive window (e.g., 5KB) is too conservative, which can lead to inferior performance, particularly for short flows. How to choose receive window for MHDs to be both resource efficient and performance enhancing is beyond the scope of this paper, but is an interesting problem that we leave as future work.

3.6.6 Application-level Protocol

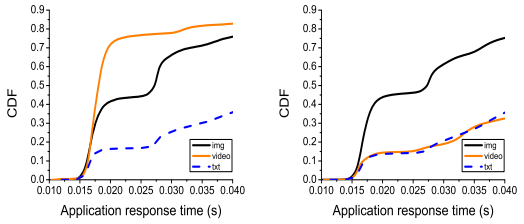
We find that some application-level protocols for MHDs are highly optimized for cellular networks, which may cause inefficient use of network and operating system resources of MHDs in WiFi networks. The native YouTube player in iOS is an example. When using this player, video contents are downloaded in segments; each segment is

requested in a separate TCP connection, using the HTTP Range header field to specify the requested portion of the video [36]. While it has been reported in literature that multiple TCP connections are used to serve a single YouTube video [36, 57], we find, surprisingly, the number of TCP connections can be extremely large. Fig. 17 shows an example where a series of 75 TCP flows (most of them very short-lived) are used to download a single YouTube video. Considering the overhead of creating new TCP connections and the slow-start phase at the beginning of a TCP connection, using so many short-lived TCP flows to serve a single video does not seem to be sensible. We suspect this is a design choice that tries to cope with TCP timeouts (e.g., caused by long delays originated from handoffs, disconnections, and MAC-level retransmission) in cellular networks [27]. For example, the handoff in cellular network can take more than one second, which might cause TCP timeout and reduced congestion window (in the worst case, the congestion window might be reduced to one). Therefore, it might make sense to open a new TCP connection to take advantage of large initial congestion window to overcome the impact of the significantly degraded throughput.

We again find the design choices taken by iOS and Android devices are different: Android devices use only one TCP flow to download a single YouTube video. How to optimize application-level protocols for MHDs, considering the characteristics of both cellular and WiFi networks, is an interesting problem, but is beyond the scope of this paper.

3.6.7 Application Response Time

We investigate application response time for different types of content to understand whether servers use different mechanisms based on content type (recall application response time represents the delay from the first GET message to the first responding

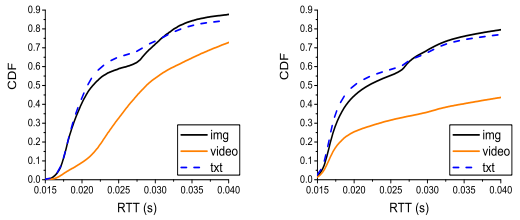


(a) MHD (b) NHD

Figure 18: Application response time for MHD and NHD flows that are served by Google servers.

data packet for an HTTP request). Fig. 18(a) plots application response time distribution for three types of contents, video, image and text, that are served by Google servers. Most of the requested videos are YouTube videos, which are destined to Google servers because Google has migrated YouTube servers to its own network infrastructure after acquiring YouTube [67]. We observe from Fig. 18(a) that, perhaps surprisingly, videos have the lowest response time, followed by images and text. The reason why videos have the lowest response time might be that users tend to watch popular videos (note that YouTube iOS application has "Featured" and "Most Viewed" categories, convenient for users to select and watch those popular videos) and servers cache popular videos, leading to low response time. The slowest response time for texts might be because most of the requests for texts query search engines, which can take a longer time to respond.

Interestingly, the results for NHDs (see Fig. 18(b)) differ from those for MHDs: for NHDs, the response times for videos and texts are similar. We do not know the exact reasons that cause the difference between MHDs and NHDs. One observation is that the user interface for NHDs differ from that in MHDs: for NHDs, YouTube suggests related videos based on a user's preference (such as browsing history, local



(a) MHD (b) NHD

Figure 19: RTT for MHD and NHD flows that are served by Google servers.

cache, etc.) while for MHDs, YouTube suggests featured and most viewed videos. Further investigation is left as future work.

In Fig. 18, we observe that the distribution of the response times for image flows that are served by Google contains two modes. We conjecture that this is because the images are served by two clusters of servers, one closer to UConn than the other. Since Google data center information is not disclosed to the public, we try to infer the Google server locations through the RTT distribution [67]. Fig. 19 presents the RTT distributions for the flows with different content types. We indeed observe that the RTT distribution for image flows contains two modes, implying that they may be served by two clusters of servers at different geographic locations.

3.6.8 Number of Concurrent TCP Flows

As stated in Section 3.5.4, the number of concurrent TCP flows might be an indicator of the amount of CPU and I/O activities on an MHD device. We next present the relationship between the number of concurrent TCP flows and per-flow servicing rate.

We find that many G_3 flows have more than one concurrent TCP flow. Fig. 20 plots the distribution of the average number of concurrent TCP flows for G_3 flows served by Akamai-113km servers (for comparison, the results for both MHDs and NHDs are

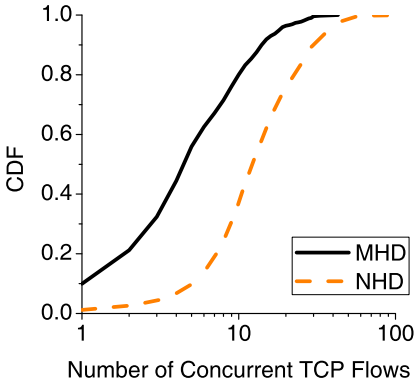


Figure 20: Average number of concurrent TCP flows for G_3 flows served by Akamai-113km servers.

plotted in the figure). For MHDs, over 90% of the G_3 flows have more than one concurrent TCP flow, and there can be tens of concurrent flows. The results for the other three server clusters are similar (figure omitted). Intuitively, a larger number of concurrent flows on an MHD may lead to less resources to each flow, and hence lower per-flow servicing rate. To verify this conjecture, we obtain the correlation coefficient between the average number of TCP flows that are concurrent with a G_3 video flow and the per-flow servicing rate of the G_3 video flow. We find that the correlation coefficients are -0.03, -0.4 and -0.29 for G_3 video flows served by Akamai-Hartford, Akamai-113km and Google servers, respectively. The negative values indicate that indeed more concurrent flows can lead to lower per-flow servicing rates. The correlation is more significant for video flows served by Akamai-113km and Google. For the flows served by Akamai-Harford, the less significant correlation might be due to the superior performance achieved by this server cluster, which makes the effect of other factors less visible.

For comparison, let us look at the distribution of the average number of concurrent TCP flows for G_3 NHD flows in Fig. 20. Not surprisingly, the number of concurrent

TCP flows on NHDs can be much larger than that on MHDs. On the other hand, we observe less significant correlation between the number of concurrent TCP flows and per-flow servicing rates on NHDs: in general, the negative correlation coefficient varies between -0.12 and -0.10 for the NHD flows served by the four server clusters. This less significant correlation may be due to superior computation and I/O capabilities on NHDs.

3.6.9 Findings from the 2012 Data Set

In the previous sections, we have presented the results from the data set collected in March 2011. We next report the results from the data set collected in April 2012.

In this data set, we find that only 64.7% of the TCP flows use HTTP, compared to 92.3% in the 2011 data set. On the other hand, the percentage of HTTPS flows has increased from 4.3% to 25.3%. This sharp increase is caused by the fact that many popular web sites such as Google, Facebook, and Hotmail, add the functionality to access their services using HTTPS, which protects users' privacy especially when they use public WiFi access points. For the rest of the TCP flows (11%), we do not observe any dominant application protocol.

Since the percentage of HTTP flows in the 2012 data set is much lower than that in the 2011 data set, and our approach to classifying MHD and NHD flows cannot be applied to the non-HTTP flows (since it uses User-Agent field in HTTP headers), leaving a large percentage (36.3%) of the TCP flows in the 2012 data set not analyzed, we have focused on the 2011 data set in this paper. We next describe the main results from analyzing the HTTP flows in the 2012 data set (developing new approaches to identify the device types for the non-HTTP flows in the 2012 data set, and analyzing

their characteristics are left as future work). In the interests of space, we only present the main findings that differ from those from the 2011 data set.

- We observe that more MHD and NHD flows are served by Akamai and Google servers. Specifically, 43.5% of MHD flows and 38.4% of NHD flows are served by these two sets of servers, respectively. The number of distinct destination host domains accessed by MHDs has increased significantly to 9.3k (compared to 6k in the 2011 data set), while for NHDs, the increase is less dramatic (from 51k to 53k). The destination host domains accessed by MHDs are still less diverse than those accessed by NHDs. This is also evidenced by the observation that for MHDs, 38% of the flows are destined to the top ten host domains, while for NHDs, the percentage is 29%.

- We find that Google has deployed a set of servers (containing 12 IP addresses) very close to UConn, in the Connecticut Education Network at Hartford. Similar to the Akamai-Hartford server, this set of close-by Google servers provide high per-flow servicing rate due to small RTTs. We find 2.8% of MHD flows are served by these servers. In addition, analyzing the content type, we find that they mainly serve Youtube traffic (account for 45% of the MHD flows from these servers).

- The amount of Android MHD flows has increased to 10% (compared to 3.2% in the 2011 data set), indicating the increasing popularity of Android devices. In addition, we observe 12% of the Android flows start with a significantly larger advertised receive window of 14KB, compared to the small initial receive window of 5KB that is observed predominantly in the 2011 data set. This larger initial receive window was adopted by newer versions of Android OS (starting from Android OS 4.X), and allows Android devices to better utilize the larger initial

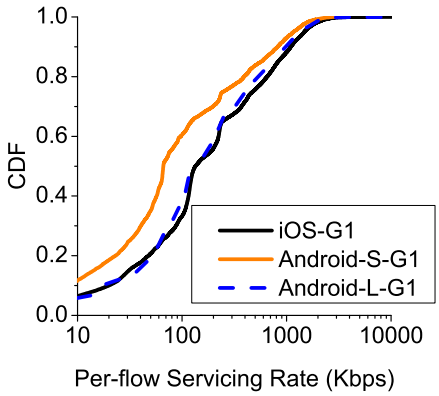


Figure 21: Distribution of per-flow servicing rate of G_1 iOS and Android flows that are served by Google servers from the 2012 data set, where Android-S-G1 corresponds to Andriod flows using small initial receive window (5KB) and Android-L-G1 corresponds to Andriod flows using large initial receive window (14KB).

congestion window adopted by many servers. Fig. 21 plots the distribution of per-flow servicing rate of iOS and Andriod flows that are served by Google servers. We separate the Android flows into two groups, one using initial receive window of 5KB, and the other using initial receive window of 14KB. We only plot the results for G_1 flows; the results for G_2 and G_3 flows are consistent. Fig. 21 shows that iOS devices outperform Android devices that use small initial receive window, a point that we made in Section 3.6.5. On the other hand, the performance of Android devices that have adopted large initial receive window is comparable to that of iOS devices.

3.7 Discussion

Our study has been conducted in a specific WiFi network, UConn campus WiFi network. A natural question is what findings from our study are specific to UConn network and what are applicable to other WiFi networks.

- Our findings related to hardware and software of MHDs and application-level protocols are not specific to UConn network, and hence we believe that they are equally applicable to other WiFi networks.

- The content delivery infrastructures in other networks may differ significantly from that perceived by UConn network. On the other hand, we believe MHDs in other networks also use Akamai and Google servers heavily because the most popular services accessed by MHDs are provided by major companies such as Facebook, Google, and Apple that use Akamai and Google servers heavily. The extent of usage and whether the usage by MHDs is heavier than that by NHDs in other networks, however, depend on the popularity of the applications in other networks.

- The use of Akamai and Google servers can also boost the network performance of MHDs in other networks in the US due to small RTTs provided by these servers. Specifically, the study of [40] reports that Akamai tends to deploy CDN sites close to the end users, and among all of the Akamai services, the 10th percentile and median of delays to Akamai are around 10ms and 20ms, respectively (these delays are comparable to the median delays of 8.5ms and 20ms from UConn campus to the two closest Akamai sites); Google is reported to provide similar RTT performance to most of the users in the US [58].

- The loss rates and RTTs in other networks may be significantly larger than those in UConn network. This implies that local losses and RTTs may play a significant role in network performance of MHDs in other networks. In addition, large initial congestion window adopted by many servers may not be strictly beneficial in other networks.

3.8 Summary

In this paper, we have studied the network performance of MHDs inside UConn campus network. We find that, compared to NHDs, MHDs use well provisioned Akamai and Google servers more heavily, which boosts the overall network performance of MHDs. Furthermore, MHD flows, particularly short flows, benefit from the large initial congestion window that has been adopted by Akamai and Google servers. Secondly, MHDs tend to have longer local delays inside the WiFi network and are more adversely affected by the number of concurrent flows. Thirdly, Android OS cannot take advantage of the large initial congestion window adopted by many servers, while the large receive window adopted by iOS is not fully utilized by most flows, leading to waste of resources. Last, some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. Our observations provide valuable insights on content distribution, server provisioning, MHD system design, and application-level protocol design.

Chapter 4

Session Length and IP Address Usage of Smart Mobile Handhelds in Wireless LANs: Characterization and Modeling

4.1 Introduction

Dynamic Host Configuration Protocol (DHCP) [31] facilitates automatic management of IP addresses in a network. It minimizes configuration errors and reduces the workload of network administrators. In addition, it is flexible: network administrators can define global and subnet-specific TCP/IP configurations, for example, they can define different settings for LAN and wireless LAN [49, 71]. The above features make DHCP the *de facto* protocol for managing IP addresses in campus and enterprise networks.

Smart mobile handheld devices (MHDs), such as iPhones, iPod touches, Android phones, Windows phones, and Blackberry phones, when connecting to wireless LANs, also obtain IP addresses through DHCP. Compared to wireless non-handheld devices

(NHDs) such as Windows laptops and MacBooks, MHDs are being adopted at a much faster pace. For instance, we observe that the number of MHDs doubled from March 2011 to March 2012 in the University of Connecticut (UConn), while the number of NHDs increased by only 20%. In addition, since MHDs are smaller and easier to carry, they tend to be more mobile, and hence have different network usage characteristics, particularly *session length*, which indicates the length of a client stays in the network. The study of session length of network clients is important for several reasons. For instance, the understanding of session length is critical for network modeling and mobility study [23, 70]. In addition, it can benefit administration, capacity planning and deployment of wireless infrastructures, protocol design for wireless applications and services, and their performance analysis. Access points, proxies, and servers can use the estimation of their clients' session lengths to prepare the handoff, share clients or traffic load with each other, and ensure a better service quality [52]. However, most existing studies focus on the session lengths of NHDs. Clearly, the more mobile nature of MHDs brings different session length characteristics. Furthermore, MHDs have higher network access frequency, and different demands on IP addresses compared to NHDs due to high network access frequency.

The above characteristics of MHDs pose the following questions on networking management in wireless LANs: *How does session length of MHDs differ from that of NHDs? How does the demand on IP addresses grow with the fast adoption of MHDs?* In this paper, we answer the above questions using a modeling approach. Specifically, we analyze two five-week long traces of two semesters for the wireless LAN at UConn, and develop hyper-exponential models for the session length of MHDs, and a model to predict the number of concurrent IP addresses at each time point in the wireless LAN.

Although motivated by MHDs, our model is also applicable to NHDs[1] . Evaluation using the five-week long trace demonstrates that our model is accurate. More specifically, the average difference of the predicted value from the model and the actual value is between 8% to 12%. To the best of our knowledge, this is the first analytical model for IP address space usage when the IP addresses are managed by DHCP. Using our model, network administrators can predict the demand on IP addresses in a wireless LAN, and take proactive actions to satisfy future IP address demands.

As related work, Khadilkar *et al.* propose emulation-based techniques to optimize DHCP lease time [49]. They do not derive analytical models as in our study. Our analytical model is helpful in optimizing DHCP lease time, and furthermore, it can be used to predict the demand on IP addresses based on the predicted changes in the number of wireless devices, which cannot be done easily using the techniques in [49]. The study in [71] develops a tool to debug DHCP performance and find DHCP misconfiguration in a campus network, which differs in scope from our study. Recently, a flurry of studies are on smartphones (e.g., [33, 37]). However, none of them is on smartphone IP address usage in wireless LANs. Papapanagiotou *et al.* study the IP leasing of smartphones, and propose dynamic IP leasing times according to minimize the DHCP server workload [53]. However, their technique cannot estimate or predict the concurrent number of users in the network. In [39, 51, 52], BiPareto distribution is proposed to model the session length. We use two and three stage hyper-exponential models to describe the session length.

The rest of the paper is organized as follows. Section 4.2 presents background on DHCP. Section 4.3 describes our data set and methodology. Section 4.4 presents our approach to model the session length distribution. Section 4.5 presents IP address

[1]In this paper, we focus on wireless devices only since the number of wireless devices is more dynamic, and can grow at a much faster speed than wired devices.

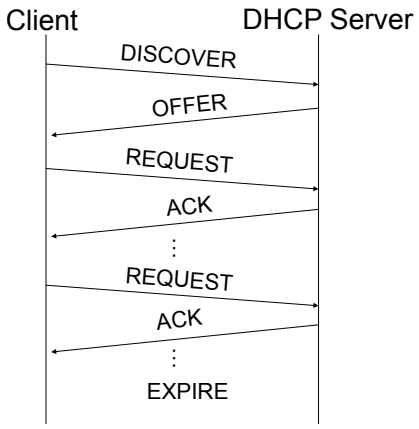


Figure 22: An example DHCP message exchange.

usage statistics and our model. Section 4.6 discusses briefly how our model can help network administrators and presents two applications of applying our model to predict the concurrent number of users in the network. Finally, Section 4.7 concludes the paper and presents future work.

4.2 Background on DHCP

We next briefly describe DHCP; more details can be found in [31]. In general, DHCP allocates an IP address from a predefined IP address pool to a host that joins the network, and reclaims that IP address when the lease time expires. Fig. 22 illustrates an example DHCP message exchange between a host and a DHCP server. When a host connects to a network, it first broadcasts a *discover* message, and each of the available DHCP servers replies with an *offer* message that contains the offered IP address[2] . Among all of the offered IP addresses, the host chooses one and sends a unicast *request* message to the DHCP server. After that, the server confirms the IP allocation via an ACK message with an IP lease time. The host needs to renew the lease by sending

[2]Most routers can forward DHCP configuration requests, eliminating the requirement of setting up a DHCP server on every subnet [6].

a *request* message again after half of the IP lease time [31, 49]. If the host does not send a *request* message before the expiration time, the server reclaims that IP address back to the pool, and logs an *expire* message. When leaving the network, the host can send an *release* message to the server (optional). An important parameter for DHCP configuration is the default IP lease time, which specifies by default, how long a host can lease an IP address. In UConn, the default IP lease time for the wireless LAN is 30 minutes.

4.3 Data Collection & Methodology

4.3.1 Data

UConn has two campus-wide DHCP servers, both running a modified version of ISC-DHCP, and logging all DHCP related messages into a central database. We obtain two sets of DHCP records from the database for Spring and Fall semesters, 2012. Each dataset contains five weeks records. One dataset is around 26 GB (February 20 to March 25, 2012), containing around 209M records. Another is around 34GB (Aug 20 to Sep 23, 2012), containing around 279M records. We refer these two data sets as Spring and Fall datasets in the following. Since UConn uses separate IP address pools for wired and wireless devices, we can extract the DHCP trace that only contains messages for wireless devices and use it for our study. Each entry in the trace corresponds to a DHCP message with MAC address, the time of the message, and other information (depending on the type of the message). We use host MAC address to uniquely identify hosts.

Within the two periods, there are two break weeks. One is between March 12 and March 18; another is between Aug 20 and Aug 26. The size of the trace for the break

week is significantly smaller (around 10% of that for a regular week). In the rest of the paper, we focus on the other eight regular weeks.

4.3.2 Methodology

The DHCP trace contains a mixture of records for MHDs and NHDs. Since MHDs and NHDs may have different characteristics in terms of when and how long they lease IP addresses (as we shall see), we first describe a methodology that determines whether a host is an MHD or NHD. We then describe a methodology to obtain IP lease information for each device. As shown in Section 4.2, the IP address will not be reclaimed until it is expired, either by IP lease time out or explicit *expire* message. Therefore, one client could own multiple IPs at one time from the DHCP server's point of view. We then describe a methodology to obtain the IP allocation information. In the rest of the paper, we use user, host, and client interchangeably.

4.3.2.1 Determining Host Type

The DHCP trace contains host name for each host (that is identified by host MAC address). We use keywords in host name to determine the type of a device. Keywords for MHDs include iPhone, iTouch, iPad, Android, Blackberry and so on; keywords for NHDs include PC, Windows, Macbook and so on. Using this approach, we can determine host types for over 90% of the hosts.

To verify the accuracy of the above method, we use an alternative approach to determine host type. Specifically, we capture one week, from March 19 to March 24, 2012, of HTTP traffic (up to 500 bytes for each packet so that HTTP header is captured), and use the user-agent field in HTTP headers to identify the OS type of each IP address [37, 48]. We then correlate the HTTP traffic trace and the corresponding

DHCP traces using the source IP address and the time so as to obtain a mapping between an IP address and an MAC address. In this way, we can obtain the OS type for each MAC address, which can be used to easily determine whether a host is an MHD or NHD. The results using this alternative approach agree with those using host names (there is only 0.39% discrepancy). In the rest of the paper, host classification is from the approach based on host names. From both DHCP traces, we identify about 48,000 unique NHDs and 18,000 MHDs.

4.3.2.2 Estimating IP Lease Information

Consider a host. Let t_s denote the time when the host starts to own an IP address, and t_e denote the time that the host does not use this address. In the rest of the paper, we refer to $t_e - t_s$ as a *session length*. From Fig. 22, we see t_s is the time of the first ACK message after the DISCOVER message, which can be easily determined from the DHCP trace. Determining t_e is much more challenging. This is because the RELEASE and EXPIRE messages are optional and indeed we find that they are not frequently used in practice. In addition, we find that a host may join the network again before the current IP lease expires. Summarizing the scenarios we observe from the DHCP trace, we propose the following rules to determine t_e.

R1: If the IP address is assigned to another host at time t, which is earlier than the lease expiration time, we set $t_e = t$.

R2: If the host sends DISCOVER message at time t, which is earlier than the lease expiration time, we set $t_e = t$.

R3: If observing an EXPIRE or RELEASE message, then t_e is the time when the message is observed.

R4: If no EXPIRE or RELEASE message is observed before the expiration time, then we use the lease expiration time as t_e[3] .

R1 corresponds to scenarios where the DHCP server assigns the IP address to another host even before the current lease expires, which should happen very rarely in a normally running network (See Table 7). R2 corresponds to scenarios where the host initiates another IP lease period before the current lease expires, and hence the current IP lease period should be terminated. R3 corresponds to scenarios where t_e is indicated by an explicit message, while R4 corresponds to scenarios where t_e is determined based on IP lease expiration time in the absence of any explicit message. The above rules are more comprehensive than those in [49] (which only consider the last two cases).

Table 7 presents the percentage of scenarios following each of the four rules described above from the Fall dataset. The Spring dataset has similar results, and the results are omitted. First, we observe for MHDs, the percentage of scenarios following R2 is significantly larger than that for NHDs, indicating that MHDs connect to the network at a much higher frequency. Second, R3 and R4 in total only account for 34.4% and 77.6% of the scenarios for MHDs and NHDs, respectively. Hence it is important to consider rules R1 and R2 in UConn network (only applying R3 and R4 as in [49] is insufficient for UConn network).

Table 7: The percentage of scenarios following each of the four rules (Fall dataset).

	R1	R2	R3	R4
MHD	0.02%	66.1%	33.1%	1.3%
NHD	0.03%	22.4%	63.4%	14.2%

[3]Since we focus on when the IP will be reclaimed, we use the expiration time instead of the last ACK message time plus half of the default renew period [49].

4.3.2.3 Estimating the IP Allocation Information

Consider an IP. Let t_a denote the time when the IP is assigned to one host, and t_b denote the time that the DHCP server reclaims the address from that host. In the rest of the paper, we refer to $t_b - t_a$ as a *IP allocation length*. Clearly, *session length* and *IP allocation length* could be different. This is because that the DHCP server does not reclaim the IP from the client until the IP leasing expires or the client sends out the *expire* message explicitly. For instance, a client obtains IP i at time t_1, and it will expire at t_2. However, that client moves to another location and obtains a different IP j at time t_3 ($t_3 < t_2$). Hence, from the DHCP server's perspective, that client owns two IPs simultaneously from t_3 to t_2.

We use R1, R3, and R4 to determine t_b. Table 8 presents the percentage of scenarios following each of the three rules. First, we observe that the large percentage of R4 indicate that most clients do not send *expire* or *release* messages to the DHCP server. This is consistent with other studies [49]. Second, 9% MHD, 3% NHD IP allocation events are terminated by R1, which means the DHCP has to reclaim some IPs even before their expiration due to unexpected events, such as IP address shortage.

Table 8: The percentage of scenarios following each of the three rules (Fall dataset).

	R1	R3	R4
MHD	9%	36%	55%
NHD	3%	20%	77%

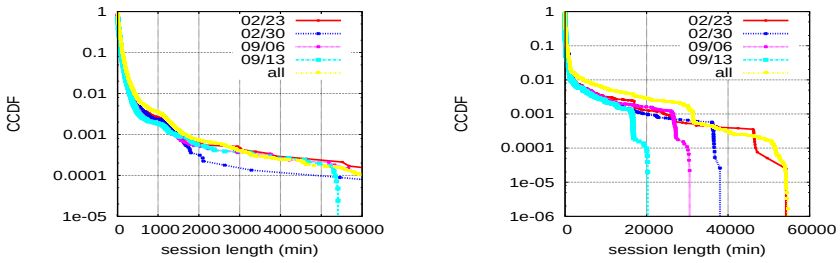
4.4 Session Length

The session length has been modeled using BiPareto distributions [39,51,52] or general distribution with certain characteristics [23]. However, BiPareto distributions focus

more on the tail part, i.e., for the extremely long session length samples. In this work, we study the IP address usage characteristics affected by the session length distribution. The long session periods take a small percentage and they don't affect the IP address usage very much. In this section, we utilize two and three stage hyper-exponential distributions to model session lengths. The hyper-exponential distributions have more control over the first and second moments. Three stage hyper-exponential distribution can even control the mediate curve. Since most of the session length periods are short, they affect the IP addresses characteristics significantly. Therefore, hyper-exponential distributions are more suitable here. Furthermore, hyper-exponential distributions are applied in high variance scenarios, which is true in this study. We next will present the empirical session length distribution and derive the model using hyper-exponential distributions. The Kolmogorov-Smirnov test are applied to evaluate the goodness of fit.

4.4.1 Session Length Distribution

For users arriving between each of the two hour intervals between 9am and 3pm, we obtain their corresponding session length distribution and find that the distributions are similar. We therefore obtain the overall session length distribution for all hosts arriving between 9am and 3pm. Figures 23(a) and (b) plot the CCDF (Complementary Cumulative Distribution Function) of session length on the four Thursdays of both Spring and Fall datasets for MHDs and NHDs, respectively. We also present session length distribution for all of the session periods in the two datasets in Figure 23. The session length distributions are similar over the two semesters. Recall that 30 minutes is the default IP lease time for UConn network. Therefore, users that stay in the network shorter than 15 minutes will own their IP addresses for 30 minutes, leading



(a) MHD (b) NHD

Figure 23: Session length distribution on the four Thursdays.

to a large percentage of session length as 30 minutes (9.4% and 10.9% for MHDs and NHDs, respectively). This causes a jump in the CCDF curve (not clearly visible in Fig. 23).

4.4.2 Models

Let $f(x)$ denote the probability density function for session length. Examining the distribution of session length (see Fig. 23), we observe that the CCDF curves can be fit by two or three lines. Therefore, we use two and three stage hyper-exponential distributions to model it. We will present the comparison of these two distributions results later. In addition, a large percentage (9-11%) of session length equals to the default IP lease time (30 minutes in UConn). We use $\delta(\cdot)$ function to model this large percentage jump. Summarizing the above, we represent $f(x)$ as

$$f(x) = (1-b)f_1(x) + b\delta(x - x_v) \tag{12}$$

where x_v is the default IP lease time, $b \in [0, 1]$ represents the fraction of session length of x_v, $\delta(\cdot)$ is the indicator function, $f_1(x)$ is a two or three stage hyper-exponential distribution.

4.4.2.1 Two Stage Hyper-exponential Distribution

We next present the modeling with two stage hyper-exponential distribution $f_1(x)$.

$$f_1(x) = \mu_1 p e^{-\mu_1 x} + \mu_2(1-p)e^{-\mu_2 x} \tag{13}$$

where $\mu_1 > 0, \mu_2 > 0$, and $p \in [0,1]$.

Let random variable X denote session length. Then the first two moments of X are

$$E(X) = (1-b)\alpha + bx_v, \tag{14}$$

$$E(X^2) = (1-b)\beta + bx_v^2, \tag{15}$$

where

$$\alpha = \frac{p}{\mu_1} + \frac{1-p}{\mu_2}, \tag{16}$$

$$\beta = \frac{2p}{\mu_1^2} + \frac{2(1-p)}{\mu_2^2}. \tag{17}$$

We can obtain $E(X), E(X^2)$ and b directly from the trace. Therefore, we can solve for α and β from equations (14) and (15). After that, we have two equations (16) and (17) and three unknowns, μ_1, μ_2, and p. Therefore, we cannot obtain μ_1, μ_2, and p directly. On the other hand, many techniques can be used to create a hyper-exponential distribution that matches the given first two moments. We use the following procedure to obtain μ_1, μ_2, and p [13].

(1) Calculate $CV^2 = \frac{\beta - \alpha^2}{\alpha^2}$.

(2) Calculate $p = \frac{1}{2}(1 - \sqrt{\frac{CV^2-1}{CV^2+1}})$. This requires $CV \geq 1$, which is true in our scenario.

(3) Set $\mu_1 = \frac{2p}{\alpha}$, and $\mu_2 = \frac{2(1-p)}{\alpha}$.

4.4.2.2 Three Stage Hyper-exponential Distribution

Three stage Hyper-exponential distribution provides another way of modeling the session length. We next describe how to model the session distribution with three stage Hyper-exponential distribution $f_1(x)$.

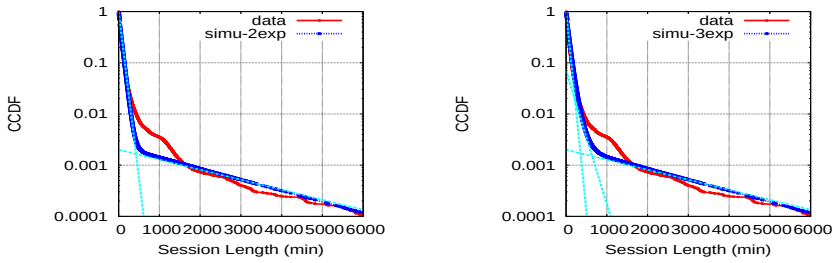
$$f_1(x) = \mu_1 p_1 e^{-\mu_1 x} + \mu_2 p_2 e^{-\mu_2 x} + \mu_3 p_3 e^{-\mu_3 x} \tag{18}$$

where $\mu_1 > 0, \mu_2 > 0, \mu_3 > 0, p_1 + p_2 + p_3 = 1$, and $p_1, p_2, p_3 \in [0, 1]$. To obtain the parameters in (18), we use the iterative Feldmann and Whitt procedure [35] as follows. Given a desired number of phases, it iteratively computes the parameters of each so as to match certain points on the CDF. The procedure starts from the tail and works its way towards the origin, taking into account the phases that have already been defined, and matching what is left over.

Let a set $0 < c_3 < c_2 < c_1$ of points divide the range of interest into exponentially-related subranges. Specifically, c_1 represents the highest values that are of interest, and c_3 represents the smallest values that are of interest. The ratio c_i/c_{i+1} is set to a constant c. We select $c = \sqrt{c_1/c_3}$. Let $q = \sqrt{c}$, where qc_1 must not be larger than the highest data point. Then, $\bar{F}(x) = Pr(X > x)$ can be calculated as follows.

1. Initially we match the first phase ($i = 1$) to the tail of the given data. In other words, we have $\bar{F}_1(x) = \bar{F}(x)$.

2. In general, in step i we match the ith phase to the tail of the remaining $\bar{F}_i(x)$. An exponential phase has two parameters, p_i and μ_i. To find the values of these two parameters, we match $\bar{F}(x)_i$ at two points: c_i and qc_i. For each phase i,



(a) Two Stage (b) Three Stage

Figure 24: Two and Three Stage Hyper-Exp matching for MHDs

$i = 1, 2,$

$$p_i = \bar{F}_i(c_i)e^{\mu_i c_i}$$
$$\mu_i = \frac{1}{(1-q)c_i} \ln \frac{\bar{F}_i(qc_i)}{\bar{F}_i(c_i)} \tag{19}$$

where

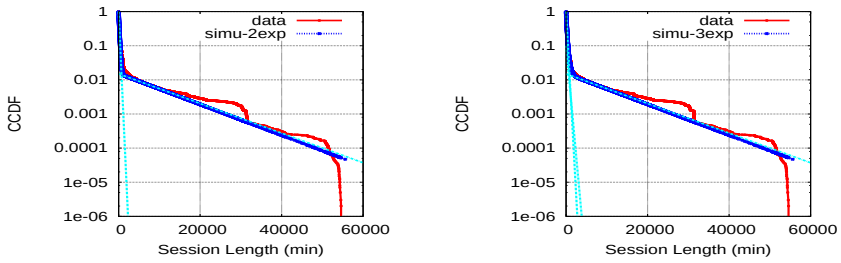
$$\bar{F}_i(c_{i+1}) = \bar{F}(c_{i+1}) - \sum_{j=1}^{i} p_j e^{-\mu_j c_{i+1}}$$
$$\bar{F}_i(qc_{i+1}) = \bar{F}(qc_{i+1}) - \sum_{j=1}^{i} p_j e^{-\mu_j qc_{i+1}}$$

3. For the last phase ($i = 3$), the procedure is different. This is to satisfy that $p_1 + p_2 + p_3 = 1$.

$$p_3 = 1 - p_1 - p_2$$
$$\mu_3 = \frac{-1}{c_3} \ln \frac{\bar{F}_3(c_3)}{p_3}$$

4.4.3 Model Validation

We next validate the model for session length distribution. Fig. 24 and 25 plot the distributions from two, three stage hyper-exponential, and the data for both MHD and



(a) Two Stage (b) Three Stage

Figure 25: Two and Three Stage Hyper-Exp matching for NHDs

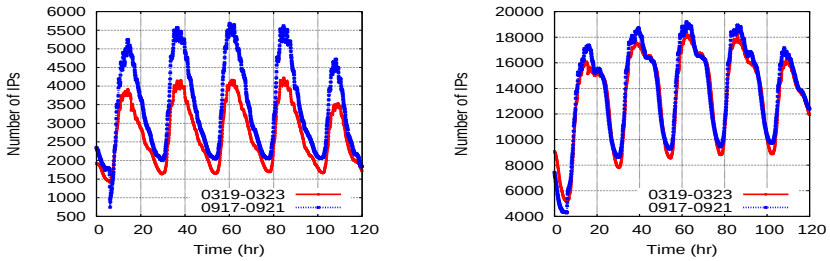
NHD data. The MHD and NHD session length distributions are obtained from 0.34M and 0.6M samples, respectively. In general, We observe a good match from both two and three stage Hyper-exponential models for both MHDs and NHDs. Furthermore, we use Kolmogorov-Smirnov test to evaluate the goodness of fit (we do not use Pearson's chi-squared test since it is very sensitive to the binning [39]). For the MHD data, the two and three stage model results are 0.08 and 0.06, respectively, indicating a good fit from both models, and three stage model slightly outperform two stage one. We observe similar results for the NHD data. The results from two and three stage models are 0.09 and 0.1, respectively.

4.5 IP Address Space Usage

We first present the weekday concurrent IPs in the WLAN. We then zoom into the peak time when most concurrent hosts appear. Last, we show the host arrival patterns.

4.5.1 Number of Concurrent IPs

Fig. 26 shows the concurrent number of IP addresses of two weeks, 3/19 to 3/23 and 9/17 to 9/21. We observe that the number of IPs is increased from March to September, and, clearly, the number of MHD clients increases at a much faster pace



(a) MHD (b) NHD

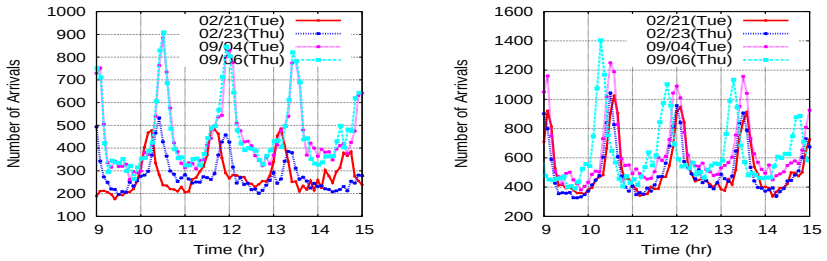
Figure 26: The number of concurrent IP addresses over a week.

than that of the NHD clients. As expected, the number of concurrent IPs follows a diurnal pattern on each day. From both datasets, we observe the largest number of concurrent IP addresses between 9am and 3pm. Furthermore, we also find that the number of concurrent IP addresses on the same weekday follows a similar trend for both MHDs and NHDs. During weekends (not plotted in the figure), the number of concurrent IP addresses is around 70% less than that on regular weekday. In this work, we more focus on the network IP space usage, especially during the peak time period (9am - 3pm). Therefore, we present the following results from the IP space usage's perspective over the peak time period.

In addition, we find the number of concurrent users on Mondays is similar to that on Wednesdays, and the number of concurrent users on Tuesdays is similar to that on Thursdays. This is because user behaviors are significantly affected by class schedules: Monday and Wednesday have similar class schedules, while Tuesday and Thursday have similar class schedules.

4.5.2 User Arrival Patterns

To measure user arrivals, we divide time into five-minute intervals, and obtain the number of user arrivals based on the number of DISCOVER messages in each interval



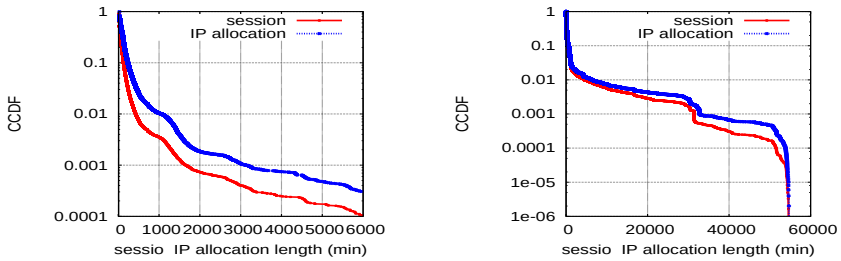
(a) MHD (b) NHD

Figure 27: The number of arrivals on February 21 (Tuesday), February 23 (Thursday), September 4 (Tuesday), and September 6 (Thursday).

(since the first DHCP message from a host that connects to a network is DISCOVER). Again, as expected, we observe that user arrivals follow a weekday pattern, and are affected by class schedules. As an example, Figures 27(a) and (b) plot the number of user arrivals from 9am to 3pm on two Tuesdays and two Thursdays for both MHDs and NHDs, respectively, from both datasets. Furthermore, in the Spring dataset, the number of MHD arrivals are much less than the NHDs. However, the difference is getting smaller in the fall dataset. As shown in Section 4.2, each client arrival causes additional DHCP server workload. Considering the fact that the concurrent number of MHD addresses in the network at one time is much less than that of the NHDs (See Fig. 26), we can see that the smaller number of MHD clients also bring significant impact to the DHCP system as the NHDs.

4.5.3 IP Allocation Length Distribution

Fig. 28 plots the session and IP allocation length distributions for both data sets. We can see the shape of session and IP allocation curves are similar. In general, the IP allocation periods are longer. This is because the user can leave the network before the ending of IP allocation period. Over the two data sets, unlike the user arrival results (See Fig. 27), the IP allocation length distributions remain the same.



(a) MHD (b) NHD

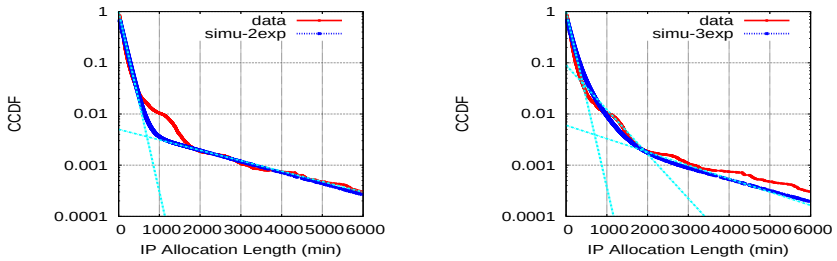
Figure 28: Session & IP Allocation length distribution.

4.5.4 Models for IP Addresses Usage

In this section, we derive an analytical model to calculate the concurrent number of IP addresses, which is applicable to both MHDs and NHDs. The concurrent number of addresses at one time is determined by two factors: the user arrival process and IP allocation length distribution. We next first present a model for user arrival process, and then a model for IP allocation length. Last, we present a model for the number of concurrent addresses, and validate the model.

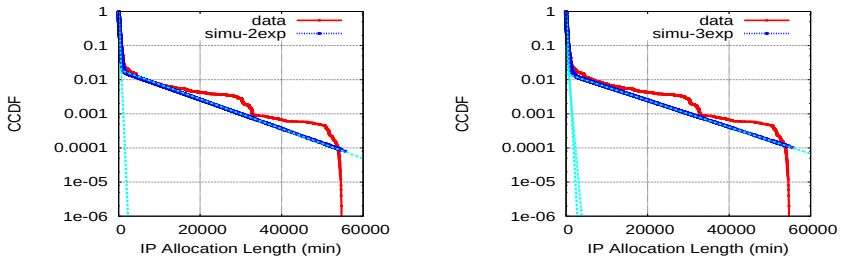
Let $\lambda(x)$ denote the user arrival rate at time x. Based on measurement results (Fig. 27), we assume that user arrival follows a Poisson distribution with a constant arrival rate over a short period of time, $[t_i, t_{i+1})$, $i = 0, \ldots, n$, and $t_0 = 0$. Then, $\lambda(x)$ can be defined as (20).

$$\lambda(x) = \begin{cases} \lambda_1 & t_0 \leq x < t_1 \\ \lambda_2 & t_1 \leq x < t_2 \\ \cdots & \\ \lambda_n & t_{n-1} \leq x < t_n \end{cases} \tag{20}$$



(a) Two Stage (b) Three Stage

Figure 29: Two and Three Stage Hyper-Exp matching for MHD IP Allocation Length Distribution



(a) Two Stage (b) Three Stage

Figure 30: Two and Three Stage Hyper-Exp matching for NHD IP Allocation Length Distribution

For ease of exposition, define $\Lambda(x) = \int_0^x \lambda(t)dt$. That is, $\Lambda(x)$ represents the total number of IP addresses that are allocated before x. Then for $x \in [t_{k-1}, t_k)$, we have

$$\Lambda(x) = \sum_{i=1}^{k-1} \lambda_i \cdot (t_i - t_{i-1}) + \lambda_k \cdot (x - t_{k-1}). \tag{21}$$

We use the same methodology as shown in Section 4.4.1 to analyze the IP allocation length (as defined in Section 4.3.2.3) distributions. Figs. 29 and 30 present the two and three stage Hyper-Exponential distributions matching the IP allocation length distribution for MHDs and NHDs, respectively. Again, three stage distribution has better matching results.

Let $N(x)$ denote the number of concurrent hosts at time x. Since the change in the number of concurrent hosts is the difference of arrivals and departures at time x,

we have

$$\frac{dN(x)}{dx} = \lambda(x) - \int_0^x \lambda(x-t)f(t)dt, \tag{22}$$

where $\int_0^x \lambda(x-t)f(t)dt$ represents the number of hosts that are leaving at time x. $f(x)$ denote the probability density function for IP allocation length. Integrating on both sides of (22) yields

$$\begin{aligned} N(x) &= \int_0^x \lambda(t)dt - \int_0^x \int_0^y \lambda(y-t)f(t)dtdy \\ &= \int_0^x \lambda(t)dt - \int_0^x f(t)dt \int_t^x \lambda(y-t)dy \\ &= \Lambda(x) - \int_0^x f(t)dt\Lambda(x-t) \\ &= \Lambda(x) - \int_0^x f(x-t)\Lambda(t)dt \end{aligned} \tag{23}$$

Substituting (12) into (23), we have

$$N(x) = \Lambda(x) - (1-b)\int_0^x f_1(x-t)\Lambda(t)dt - b\int_0^x \delta(x-t-x_v)\Lambda(t)dt \tag{24}$$

In (24), the third term on the right hand side can be calculated as

$$b\int_0^x \delta(x-t-x_v)\Lambda(t)dt = \begin{cases} 0 & , \quad x < x_v \\ b\Lambda(x-x_v) & , \quad x \geq x_v \end{cases} \tag{25}$$

Substituting (21) into the integral part of the second term (right hand side of (24)) yields

$$\int_0^x f_1(x-t)\Lambda(t)dt = \sum_{i=1}^{k-1} \int_{t_{i-1}}^{t_i} f_1(x-t)\Lambda(t)dt + \int_{t_{k-1}}^x f_1(x-t)\Lambda(t)dt \tag{26}$$

For two stage hyper-exponential model, (26) can be obtained by calculating the integral between $[t_{i-1}, t_i)$ as

$$\begin{aligned} \int_{t_{j-1}}^{t_j} f_1(x-t)\Lambda(t)dt = &\int_{t_{j-1}}^{t_j} \mu_1 p e^{-\mu_1(x-t)}(\Lambda(t_{j-1}) + \lambda_j(t-t_{j-1}))dt + \\ &\int_{t_{j-1}}^{t_j} \mu_2(1-p)e^{-\mu_2(x-t)}(\Lambda(t_{j-1}) + \lambda_j(t-t_{j-1}))dt \end{aligned} \tag{27}$$

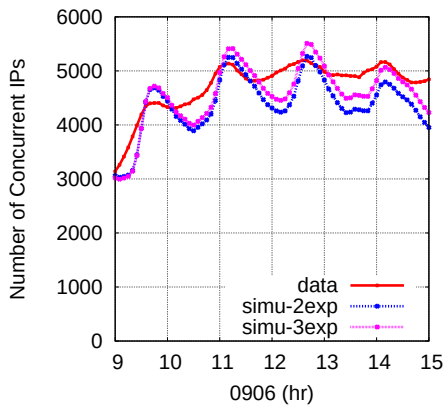


Figure 31: Compare the results from two, three stage hyper-exponential and empirical data for MHDs.

The three stage hyper-exponential model uses the following equation

$$\begin{aligned}\int_{t_{j-1}}^{t_j} f_1(x-t)\Lambda(t)dt = &\int_{t_{j-1}}^{t_j} \mu_1 p_1 e^{-\mu_1(x-t)}(\Lambda(t_{j-1}) + \lambda_j(t-t_{j-1}))dt + \\ &\int_{t_{j-1}}^{t_j} \mu_2 p_2 e^{-\mu_2(x-t)}(\Lambda(t_{j-1}) + \lambda_j(t-t_{j-1}))dt + \\ &\int_{t_{j-1}}^{t_j} \mu_2 p_3 e^{-\mu_3(x-t)}(\Lambda(t_{j-1}) + \lambda_j(t-t_{j-1}))dt\end{aligned} \tag{28}$$

Combining (24), (25), (26), and (27 or 28), we obtain $N(x)$, the number of concurrent hosts in the network. In this model, we treat $N(0) = 0$. This is because if time 0 is chosen far away from the peak time, then based on session length distribution, hosts that stay in the network at time 0 would have already left the network by the peak time.

4.5.5 Model Validation

We now validate the model for obtaining the number of concurrent IP addresses. The model needs the IP allocation length distribution and host arrival rate as inputs. As shown in Fig. 23, the session length and IP allocation length distributions are similar

over different days. However, the host arrival rate varies day to day (See Fig. 27). Hence, to validate the model, we use each day's host arrival rate and the general session length distribution as the model input to obtain the number of concurrent users. For example, Fig. 31 plots the results of MHDs from two and three stage hyper-exponential models for September 6. We observe a good fit from both models, and again, three stage model provides better matching results. Fig. 32 shows the results of NHDs from the three stage hyper-exponential model and the empirical data for September 6. In Fig. 32, we use two types of arrival rate information to calculate the number of concurrent users. One is considering the arrival rate information of September 6 only; another is considering both September 5 and 6's arrival rate data. We observe that using two days' arrival rate generates better results than using one day's arrival rate only. This is caused by the fact that around 5% NHDs staying in network for more than one day (See Fig. 23(b)). We cannot ignore these users when analyzing NHDs. MHDs stay in the network for shorter periods of time than NHDs, and hence, using one day's arrival rate as input is sufficient. Furthermore, to quantify how well our model matches the actual number of concurrent users, we calculate the relative difference (we use the absolute value) between the model and the data every 5 minutes in each day, and use the average relative difference during the entire period (9am - 3pm) as the performance metric. For all the days we examine, the average relative difference is between 8% to 12%, indicating a good match.

4.6 Applications of the IP Address Usage Model

Our model is useful for network administrators to predict the demand on IP addresses in a wireless LAN. We next present two applications of this model.

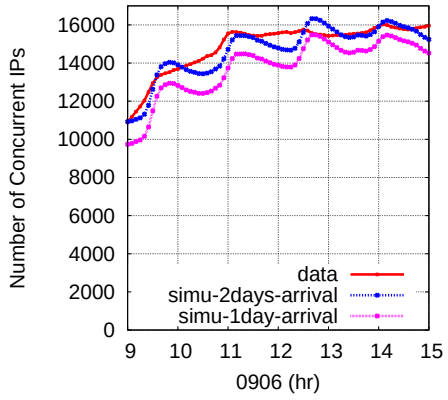


Figure 32: Compare the results from empirical data and three stage hyper-exponential with one day and two days arrival information.

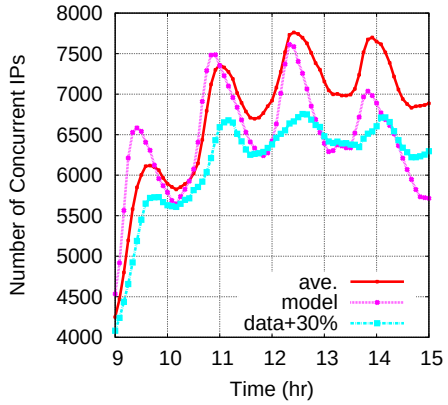


Figure 33: Compare the results from three stage hyper-exponential model and the average of simulation with 30% more arrivals of MHD clients.

4.6.1 Case Study 1

In this case study, we use three stage hyper-exponential IP allocation length model to predict the number of concurrent users in the network, based on the expected growth in the number of MHDs and NHDs. Suppose the university is going to host a big event from 9am to 3pm with large number of new participants. We assume the arrival rate of MHD clients during that period would be increased by a certain percentage. At each time point, we assume the increased percentage is uniformly chosen from 20% to

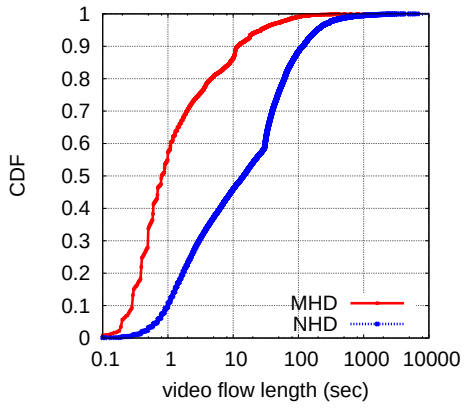


Figure 34: Video Flow Length

40% with an average of 30%. For the newly joined users, we assume they still follow the existing IP allocation distribution. Hence, we uniformly generate one IP allocation period for each new user from the existing IP allocation distribution. After that, we recalculate the concurrent number of users from 9am to 3pm. We also obtain the number of concurrent users by applying the three stage hyper-exponential model with 30% increase in host arrival rate. Another intuitive approach to estimate the concurrent numbers under this scenario is just increasing the existing number of concurrent users data by 30%. Fig. 33 presents the results from the model prediction, simulation, and this intuitive approach. The average curve is obtained by running 1000 times simulations, and the confidence interval is very tight, hence omitted. We observe that the model gives better estimation than the intuitive approach. The intuitive approach always underestimates the average concurrent users. This is because adding 30% to the existing data ignores the new users that stay in the network longer. The difference, as shown in Fig. 33, increases over time.

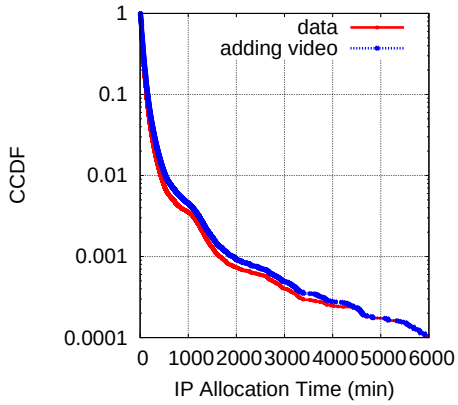


Figure 35: IP Lease Distributions by adding video watching time.

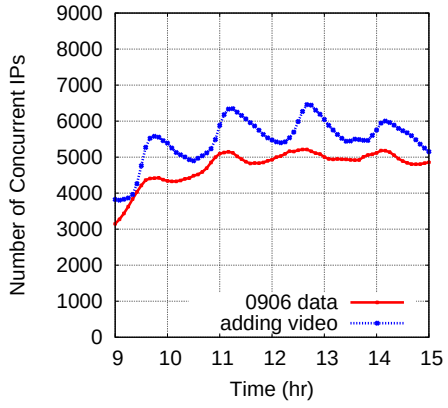


Figure 36: Predict the IP address demand with more video watching time.

4.6.2 Case Study 2

Utilizing this approach, the network administrators can also predict the IP address demand if the network usage behavior changes in the future. To study the user network usage characteristics, we capture the TCP flow traffic within UConn campus [21]. For example, Fig. 34 presents the video TCP flow length distribution for both MHDs and NHDs. On average, the MHD and NHD video flows are 6.6s and 53.8s, respectively. As online videos are getting more and more popular, users would spend more time online watching videos. We next present another case study of applying this model to predict the IP address demand when the user network usage behavior is changed. Under this scenario, we assume users will spend more time on video watching, and hence, use the network longer. To that end, for each IP allocation period, we first generate a certain number of video sessions. And then, we add these video sessions to the existing IP allocation period. The number of video sessions is uniformly chosen from 0 to 300, and each video session is randomly selected from the existing video flow length distribution (See Fig. 34). Then, we can apply this model to predict the IP address demand under this case. Fig. 36 shows the results for MHD clients. We can see that the peak IP address demand is increased as large as 25%. This indicates that the IP address demand can be increased significantly when MHDs use the network longer.

4.7 Conclusion

In this chapter, we have analyzed two five-week long DHCP traces for UConn wireless LAN, modeled the session length, and developed a model to estimate the number of concurrent IP addresses in the network. Evaluation results demonstrate

that our model is accurate. Our model can help network administrators to predict the demand on IP addresses.

Chapter 5

Conclusions and Future Works

We have presented the motivation and challenges of running fault and performance monitoring in wireless LANs and provided solutions for three different problems. In this chapter, we will conclude our work and present future works.

Wireless LANs are ubiquitous and widely used in a lot of applications. However, they are vulnerable to failures or low performance because of the limitation of their the nature such as signal fading, channel interference and collision. Furthermore, the flourish of new wireless clients, such as smart phones, poses new challenges for reliable and efficient wireless LANs. The requirements for wireless network fault and performance monitoring are from network, clients, and network management. In this dissertation, we study three distinct fault and performance measurement problems. We address the problems from three aspects: (1) assign minimum number of channels to sniffers to maintain a full monitoring network; (2) analyze the wireless LAN traffic and obtain the leading factors that could affect the network performance of MHDs; (3) analyze

the DHCP traces and use a modeling approach to estimate the number of concurrent users in the network. Our work is summarized as follows.

- *Sniffer Channel Selection for Monitoring Wireless LANs* We studied sniffer channel selection for monitoring WLANs. In particular, we formulated min-max and min-sum sniffer channel selection problems, and proposed three algorithms, one based on IP, one based on LP-relaxation, and the third based on a greedy heuristic, to solve each problem. Through simulation, we demonstrated that for each problem, all the algorithms are effective in achieving their optimization goals, and overall, the LP-based algorithm outperform the other two algorithms.

  As future work, we plan to investigate dynamic sniffer channel assignment [65] that adjusts the channel assignment in the face of faults or attacks. Furthermore, our min-max and min-sum problems consider the number of channels that a sniffer monitors as the workload of the sniffer. Another direction of future work is using the amount of traffic that a sniffer monitors as the workload.

- *Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network* In this work, we have studied the network performance of MHDs inside UConn campus network. We find that, compared to NHDs, MHDs use well provisioned Akamai and Google servers more heavily, which boosts the overall network performance of MHDs. Furthermore, MHD flows, particularly short flows, benefit from the large initial congestion window that has been adopted by Akamai and Google servers. Secondly, MHDs tend to have longer local delays inside the WiFi network and are more adversely affected by the number of concurrent flows. Thirdly, Android OS cannot take advantage of the large initial congestion window adopted by many servers, while the large receive window adopted

by iOS is not fully utilized by most flows, leading to waste of resources. Last, some application-level protocols cause inefficient use of network and operating system resources of MHDs in WiFi networks. Our observations provide valuable insights on content distribution, server provisioning, MHD system design, and application-level protocol design.

As future work, we plan to use active controlled experiments to understand why local RTTs of MHDs tend to be larger than those on NHDs, and to understand the bottleneck(s) of MHD flows. We also plan to study network performance of MHDs in other public WiFi networks. For instance, we believe that WiFi hotspots (e.g., hotspots in Starbucks) and other campus WiFi networks might have different network characteristics (e.g., higher loss rates and higher RTTs) as well as different content popularity among users. Furthermore, the content delivery infrastructure may differ significantly from that perceived by UConn network. Quantifying the network performance of MHDs and identifying the performance limiting factors in a wide range of settings will provide us better insights on designing network services for MHDs. Last, we plan to study how the performance of 3G cellular network differs from that of WiFi in our campus network and when/why users on campus switch from one network interface to another network interface.

- *Session Length and IP Address Usage of Smart Mobile Handhelds in Wireless LANs: Characterization and Modelings* In this paper, we have analyzed two five-week long DHCP traces from UConn wireless LAN, modeled the session length, and developed a model to estimate the number of concurrent IP addresses in the

network. Evaluation results demonstrate that our model is accurate. Our model can help network administrators to predict the demand on IP addresses.

As future work, we will further explore how to predict session length distribution if DHCP parameters, e.g., default IP lease time, are changed. We will also investigate how to optimize DHCP configurations using the model.

Bibliography

[1] http://csrc.nist.gov/publications/nistpubs/800-153/sp800-153.pdf.

[2] *AirDefense, Wireless LAN Security.* http://airdefense.net.

[3] *AirMagnet.* http://www.airmagnet.com.

[4] *AirWave, AirWave Management Platform.* http://airwave.com.

[5] Alexa. http://www.alexa.com.

[6] Benefits of DHCP. http://technet.microsoft.com/library/cc958943.aspx.

[7] *Cisco Wireless LAN Solution Engine (WLSE).* http://www.cisco.com/en/US/products/sw/cscowork /ps3915/.

[8] Cplex. http://www.ilog.com/products/cplex/.

[9] DAG card. http://www.endace.com.

[10] HTTP Pipelining. http://en.wikipedia.org/wiki/HTTP_pipelining.

[11] *NetStumbler.* http://www.netstumbler.com.

[12] A. Adya, V. Bahl, R. Chandra, and L. Qiu. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *Proc. of ACM MobiCom*, September 2004.

[13] A. O. Allen. *Probability, Statistics, and Queuing Theory with Computer Science Applications*. Academic Press, 1978.

[14] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window, 2002. RFC 3390.

[15] P. Arora, C. Szepesvari, and R. Zheng. Sequential learning for optimal monitoring of multi-channel wireless networks. In *Proc. of IEEE INFOCOM*, 2011.

[16] C. Arun, N. Huy, S. Gabriel, and Z. Rong. On quality of monitoring for multi-channel wireless infrastructure networks. In *Proc. of ACM MobiHoc*, 2010.

[17] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. Enhancing the security of corporate Wi-Fi networks using DAIR. In *Proc. of ACM MobiSys*, 2006.

[18] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of ACM IMC*, 2009.

[19] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *Proc. of ACM Ubicomp*, 2007.

[20] R. Chandra, J. Padhye, A. Wolman, and B. Zill. A location-based management system for enterprise wireless LANs. In *Proc. of Networked Systems Design & Implementation (NSDI)*, 2007.

[21] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network performance of smart mobile handhelds in a university campus wifi network. In *Proc. of IMC*, 2012.

[22] X. Chen, B. Wang, K. Suh, and W. Wei. Passive online wireless LAN health monitoring from a single measurement point. *ACM Mobile Computer Comm. Review*, 14, November 2010.

[23] Y.-C. Chen, J. Kurose, and D. Towsley. A mixed queueing network model of mobility in a campus wireless network. In *Proc. of IEEE INFOCOM*, 2012.

[24] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A. C. Snoeren, S. Savage, and G. M. Voelker. Automating cross-layer diagnosis of enterprise wireless networks. In *Proc. of ACM SIGCOMM*, Kyoto, Japan, August 2007.

[25] Y.-C. Cheng, J. Bellardo, P. Benko, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proc. of ACM SIGCOMM*, Pisa, Italy, September 2006.

[26] A. Chhetri and R. Zheng. WiserAnalyzer: A passive monitoring framework for WLANs. In *Proc. of Mobile Ad-hoc and Sensor Networks*, 2009.

[27] C. M. Choon and R. Ram. Improving TCP/IP performance over third-generation wireless networks. *IEEE Transactions on Mobile Computing*, 2008.

[28] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3G and metro-scale WiFi for vehicular network access. In *Proc. of ACM IMC*, 2010.

[29] U. Deshpande, T. Henderson, and D. Kotz. Channel sampling strategies for monitoring wireless networks. In *Proc. of the Second Workshop on Wireless Network Measurements*, Boston, MA, April 2006.

[30] U. Deshpande, C. McDonald, and D. Kotz. Coordinated sampling to improve the efficiency of wireless network monitoring. In *Proceedings of the Fifteenth IEEE International Conference on Networks (ICON)*, November 2007.

[31] R. Droms. Dynamic host configuration protocol, 1997. RFC 2131.

[32] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM CCR*, 40:26–33, June 2010.

[33] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proc. of ACM IMC*, 2010.

[34] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proc. of ACM MobiSys*, 2010.

[35] A. Feldmann. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. pages 245–279, 1998.

[36] A. Finamore, M. Mellia, M. Munafo, and S. G. Rao. YouTube everywhere: Impact of device and infrastructure synergies on user experience. In *Proc. of ACM IMC*, 2011.

[37] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus WiFi networks. In *Proc. of PAM*, 2011.

[38] D. S. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. *D. S. Hochbaum editor, Approximation algorithms for NP-hard problems*, pages 94–143, 1996. PWS Publishing Co. Boston, MA, USA.

[39] W.-J. Hsu and A. Helmy. On nodal encounter patterns in Wireless LAN traces. *IEEE Transactions on Mobile Computing*, 2010.

[40] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and evaluating large-scale CDNs. Technical Report MSR-TR-2008-106, Microsoft Research, 2008.

[41] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proc. of ACM Mobisys*, 2010.

[42] IP2Location. http://www.ip2location.com.

[43] R. Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[44] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. of IEEE INFOCOM*, 2004.

[45] A. P. Jardosh, K. N. Ramachandran, and K. C. Almeroth. Understanding link-layer behavior in highly congested IEEE 802.11b wireless networks. In *Proc. of ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, 2005.

[46] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the MAC-level behavior of wireless networks in the wild. In *Proc. of ACM SIGCOMM*, Pisa, Italy, September 2006.

[47] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband Internet traffic. In *Proc. of ACM IMC*, 2009.

[48] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Proc. of PAM*, 2010.

[49] K. Manas, F. Nick, S. Matt, and C. Russ. Usage-based DHCP lease time optimization. In *Proc. of ACM IMC*, 2007.

[50] A.-F. Mohammad, E. Khaled, R. Benjamin, and G. Igor. Overclocking the Yahoo!: CDN for faster web page loads. In *Proc. of ACM IMC*, 2011.

[51] C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A compound model for tcp connection arrivals for lan and wan applications. *Computer Networks*, 40(3):319 – 337, 2002.

[52] M. Papadopouli, H. Shen, and M. Spanakis. Characterizing the duration and association patterns of wireless access in a campus. *Wireless Conference 2005 - Next Generation Wireless and Mobile Communications and Services (European Wireless), 11th European*, 2005.

[53] I. Papapanagiotou, E. M. Nahum, and V. Pappas. Configuring dhcp leases in the smartphone era. In *Proc. of IMC*, 2012.

[54] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP revisited: a fresh look at TCP in the wild. In *Proc. of ACM IMC*, 2009.

[55] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proc. of ACM MobiSys*, 2010.

[56] A. Rahmati and L. Zhong. Human battery interaction on mobile phones. *Elsevier Pervasive and Mobile Computing Journal*, 5(5), 2009.

[57] A. Rao, Y.-S. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. of ACM CoNext*, 2011.

[58] K. Rupa, M. H. V., S. Sridhar, J. Sushant, K. Arvind, A. Thomas, and G. Jie. Moving beyond end-to-end path information to optimize CDN performance. In *Proc. of ACM IMC*, 2009.

[59] A. Shane and N. Richard. Application flow control in YouTube video streams. *SIGCOMM Computer Comm. Review*, 41, 2011.

[60] Y. Sheng, G. Chen, H. Yin, K. Tan, U. Deshpande, B. Vance, D. Kotz, A. Campbell, C. Mcdonald, T. Henderson, and J. Wright. MAP: a scalable monitoring system for dependable 802.11 wireless networks. *IEEE Wireless Communications*, 15(5), 2008.

[61] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell. Detecting 802.11 MAC layer spoofing using received signal strength. In *Proc. of IEEE INFOCOM*, April 2008.

[62] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. C. Sicker. MOJO: A distributed physical layer anomaly detection system for 802.11 WLANs. In *Proc. of ACM MobiSys*, pages 191–204, 2006.

[63] D.-H. Shin and S. Bagchi. Optimal monitoring in multi-channel multi-radio wireless mesh networks. In *Proc. of ACM MobiHoc*, 2011.

[64] D.-H. Shin, S. Bagchi, and C.-C. Wang. Distributed online channel assignment toward optimal monitoring in multi-channel wireless networks. In *Proc. of IEEE INFOCOM, mini conference*, 2012.

[65] R. Shravan, S. Vivek, B. Suman, and C. Ranveer. Fluid: improving throughputs in enterprise wireless lans through flexible channelization. In *Proc. of ACM MobiCom*, 2011.

[66] A. Shye, B. Sholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimization for mobile architectures. In *Proc. of IEEE/ACM MICRO*, 2009.

[67] R. Torres, A. Finamore, J. Kim, M. Mellia, M. Munafo, and S. Rao. Dissecting video server selection strategies in the YouTube CDN. In *Proc. IEEE ICDCS*, 2011.

[68] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Measuring serendipity: connecting people, locations and interests in a mobile 3G network. In *Proc. of ACM IMC*, 2009.

[69] F. P. Tso, J. Teng, W. Jia, and D. Xuan. Mobility: a double-edged sword for HSPA networks: a large-scale test on Hong Kong mobile HSPA networks. In *Proc. of ACM MobiHoc*, 2010.

[70] C. Tuduce and T. Gross. A mobility model based on wlan traces and its validation. In *Proc. of IEEE INFOCOM*, 2005.

[71] B. Vladimir, S. Jesse, and B. Suman. Debugging DHCP performance. In *Proc. of ACM IMC*, 2004.

[72] W. Wei, S. Jaiswal, J. Kurose, and D. Towsley. Identifying 802.11 traffic from passive measurements using iterative Bayesian inference. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2005.

[73] W. Wei, S. Jaiswal, J. Kurose, and D. Towsley. Identifying 802.11 traffic from passive measurements using iterative Bayesian inference. In *Proc. of IEEE INFOCOM*, 2006.

[74] W. Wei, K. Suh, B. Wang, Y. Gu, J. Kurose, and D. Towsley. Passive online rogue access point detection using sequential hypothesis testing with TCP ACK-pairs. In *Proc. of IMC*, October 2007.

[75] B. Yan and G. Chen. Model-based fault diagnosis for ieee 802.11 wireless LANs. In *Proc. of the International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*, March 2009.

[76] B. Yan, G. Chen, J. Wang, and H. Yin. Robust detection of unauthorized wireless access points. *ACM/Springer Mobile Networks and Applications (MONET)*, 14(4), August 2009.

[77] J. Yeo, M. Youssef, and A. Agrawala. A framework for wireless LAN monitoring and its applications. In *WiSe*, 2004.

[78] J. Yeo, M. Youssef, T. Henderson, and A. Agrawala. An accurate technique for measuring the wireless side of wireless networks. In *Proc. of USENIX/ACM Workshop on Wireless Traffic Measurements and Modeling (WiTMeMo)*, 2005.

[79] D. Zhang. Web content adaptation for mobile handheld devices. *Communications of the ACM*, 50(2), February 2007.

[80] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *Proc. of ACM MobiSys*, 2010.